Ein Markt & Technik Buch

HAPPY-COMPUTER

MSX-BASIC

Ein Spiel- und Lernbuch

Haupteigenschaften · Programmierung · Farbgrafik
Bewegte Bilder · Spielanweisung · 3 vollständige
Spiele · Befehlsübersicht

Joseph P. Lau · Robin Y. Law · Michael K. Wan

MSX-BASIC

MSX von Microsoft (MSX = Microsoft extended BASIC) hat sich allmählich zu einem Heimcomputer-Standard entwickelt. Zusätzlich zum »normalen« BASIC können mit insgesamt mehr als 150 Befehlen und Funktionen Grafiken erstellt, Töne erzeugt, Melodien komponiert und ganze Spielhandlungen programmiert werden. 32 Sprites garantieren abwechslungsreiche Action-Spiele.

Da sich der MSX-Standard auch auf die Hardware bezieht (Z80-CPU, 8255-PIO, identische Video- und Soundchips), kann Software auf alle MSX-Computer problemlos übertragen werden.

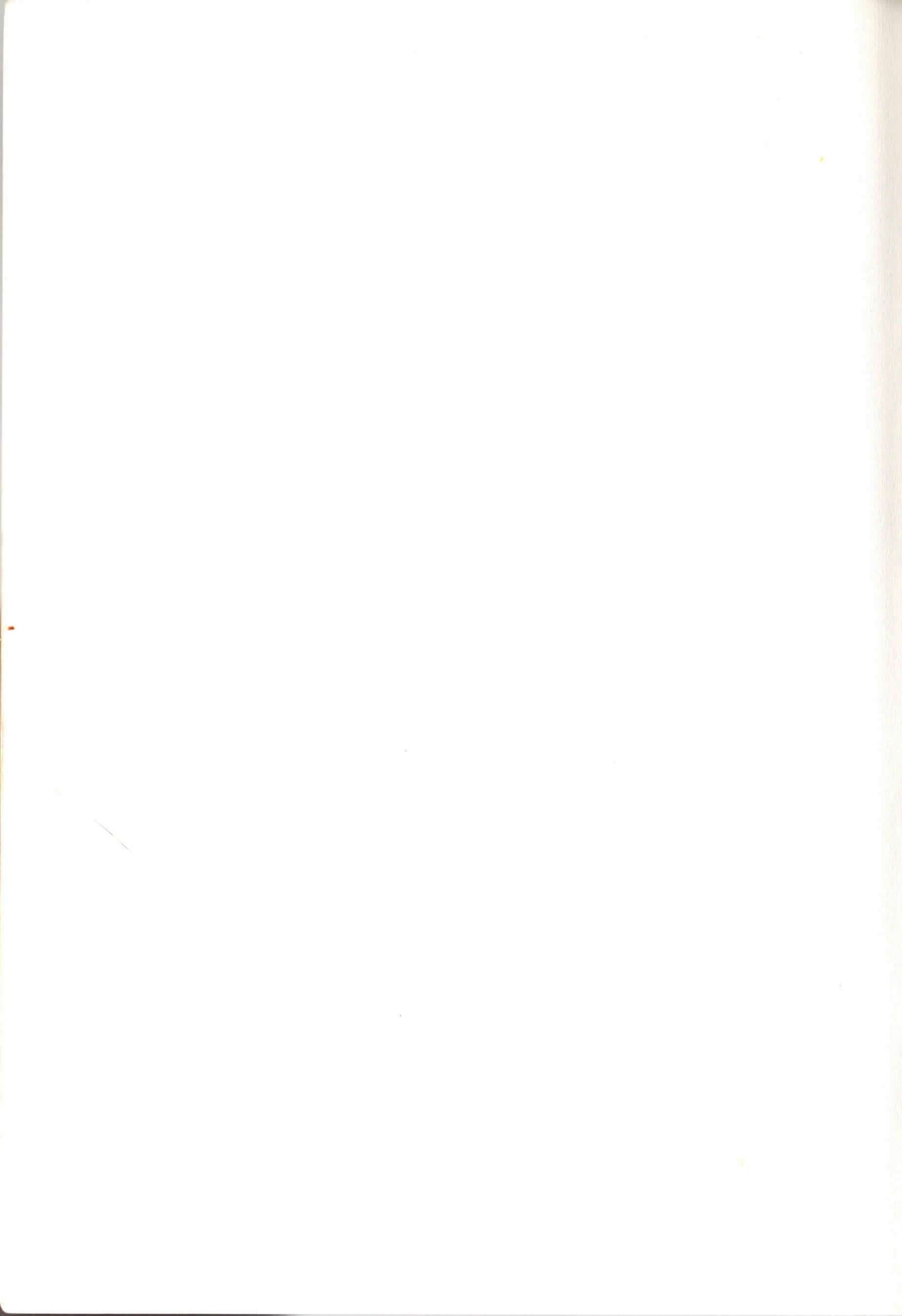
Nach einer kurzen Vorstellung der Hardware des MSX-Sytems gibt das Buch Hinweise zur Dateibehandlung (Laden und Speichern von Programmen). Anschließend lernt man anhand der Entwicklung eines Spielszenarios das MSX-BASIC kennen. Jeder neue Abschnitt baut auf dem Wissensstand des vorhergehenden auf. Auf diese Weise werden nach und nach die wichtigsten und interessantesten Befehle vermittelt.

Die letzten Kapitel des Buchs stellen drei vollständige Spiele vor, die dem neugierigen Leser die Gelegenheit geben, »hinter die Kulissen« zu sehen. Auf jeden Fall bieten die Spiele Abwechslung und Unterhaltung. Eine Kassette mit allen Programm-Modulen und den Spielprogrammen ist beim Verlag gesondert erhältlich.

Markt&Technik Verlag Aktiengesellschaft
ISBN 3-89090-107-7

DM 44,—
Sfr. 40,50 / öS 343,20

MSX-BASIC
Ein Spiel- und Lernbuch



Joseph P. Lau
Robin Y. Law
Michael K. Wan

MSX-BASIC
Ein Spiel- und Lernbuch

Haupteigenschaften · Programmierung
Farbgrafik · Bewegte Bilder
Spielanweisung · 3 vollständige
Spiele · Befehlsübersicht

Deutsche Übersetzung:
Irene Lüke

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Lau, Joseph P.:
MSX BASIC — ein Spiel- und Lernbuch : Haupteigenschaften, Programmierung, Farbgrafik, bewegte Bilder, Spielanweisung, 3 vollst. Spiele, Befehlsübersicht / Joseph P. Lau ; Robin Y. Law ; Michael K. Wan. Dt. Übers.: Irene Lüke. — Haar bei München : Markt-und-Technik-Verlag, 1985.
(Happy Computer)
Einheitssacht.: A tutorial on MSX BASIC — game approach <dt.>
ISBN 3-89090-107-7
NE: Law, Robin Y.:; Wan, Michael K.:

Titel der englischen Originalausgabe
»A TUTORIAL ON MSX BASIC — GAME APPROACH«
by Joseph P. Lau, Robin Y. Law and Michael K. Wan

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

89 88 87 86 85

ISBN 3-89090-107-7

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder, Gersthofen
Printed in Germany

Vorwort

Dieses Buch richtet sich an drei Gruppen von Lesern:

1. an Anfänger, die die Programmiersprache BASIC lernen wollen,

2. an fortgeschrittene Programmierer, die alles über MSX lernen wollen und

3. an Programmierer, die lernen wollen, wie man Actionspiele schreibt.

Wahrscheinlich ist es deshalb unvermeidbar, daß der eine oder andere Leser einen Teil des Buchs weniger interessant finden und sich nur bestimmte Stellen herauspicken wird. Damit jeder dieses Buch so wirkungsvoll wie möglich einsetzen kann, haben wir ein Flußdiagramm entworfen. Darin können Sie den Pfad wählen, der Ihrem Wissensstand und Ihrer Erfahrung entspricht und gelangen auf effektivste Weise an Ihr gewünschtes Ziel.

Für Leser, die sich das Eintippen der Programme ersparen wollen, ist eine Programmbeispielkassette erhältlich, auf der alle im Buch behandelten Programm-Module vorhanden sind. Außerdem befinden sich noch drei vollständige Spielprogramme darauf, die den Einstieg in die MSX-Computerwelt erleichtern sollen.

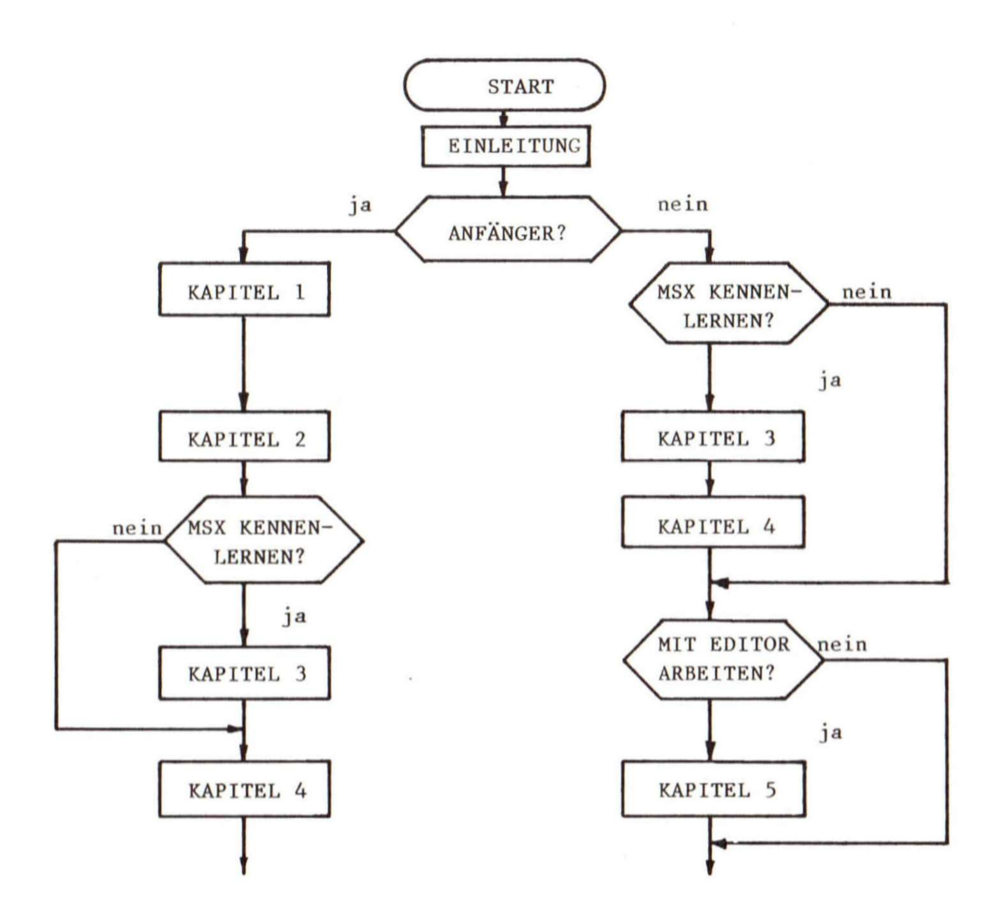


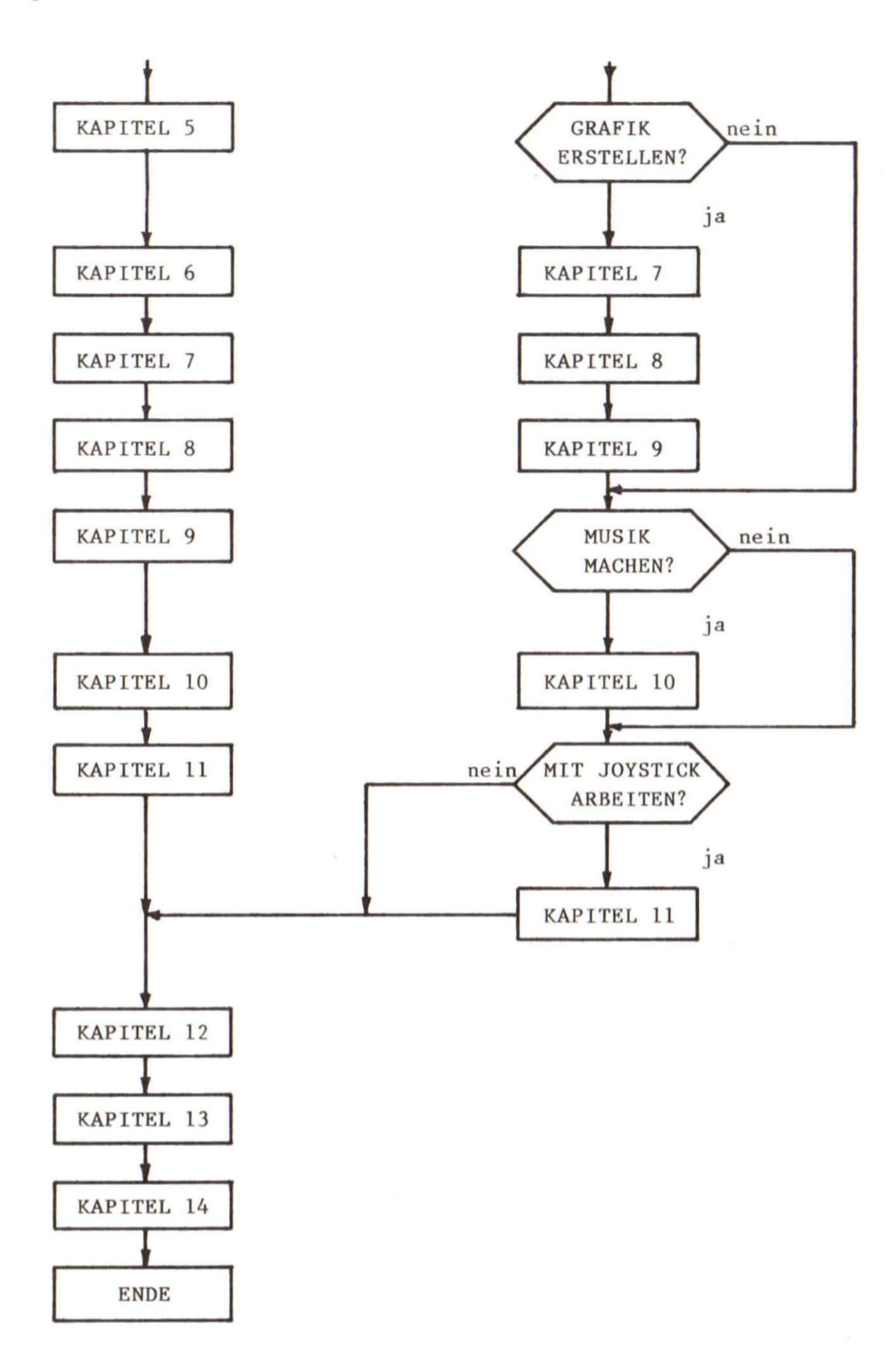
Inhaltsverzeichnis

Einleitung

Das Flußdiagramm auf den nächsten Seiten hilft Ihnen, dieses Buch möglichst effektiv einzusetzen. Doch zuerst einige Hinweise:

Die Zahl innerhalb der Rechtecke repräsentiert die Kapitelnummer. Beginnen Sie das Flußdiagramm an der Marke START. Folgen Sie dem Pfad, bis Sie zur ersten Weggabelung kommen, die durch ein Sechseck dargestellt ist. An dieser Stelle müssen Sie den Weg einschlagen, der durch die für Sie zutreffende Bedingung freigegeben wird. Setzen Sie ihren Weg fort, bis Sie schließlich das Ende erreichen. Dabei müssen Sie weitere Entscheidungen treffen. Sie sehen, der Lesestoff hängt also von dem von Ihnen gewählten Pfad ab.




KAPITEL 5
KAPITEL 6
KAPITEL 7
KAPITEL 8
KAPITEL 9
KAPITEL 10
KAPITEL 11
KAPITEL 12
KAPITEL 13
KAPITEL 14
ENDE
GRAFIK ERSTELLEN?
nein
ja
KAPITEL 7
KAPITEL 8
KAPITEL 9
MUSIK MACHEN?
nein
ja
KAPITEL 10
MIT JOYSTICK ARBEITEN?
nein
ja
KAPITEL 11

Wenn Sie sich die Arbeit erleichtern wollen, können Sie eine für dieses Buch erstellte Programmkassette erwerben, die alle Programm-Module (das sind die Programmsegmente) beinhaltet, die im Buch behandelt werden. Im Anhang D finden Sie das Inhaltsverzeichnis der auf der Kassette vorhandenen Programme. Sie können jetzt dem Buch gemäß jedes Programm-Modul ausdrucken oder Sie können auch die entsprechenden Programm-Module laden und sie laufen lassen. Sie müssen selbst entscheiden, bei welcher Vorgehensweise sie sich den größten Erfolg versprechen.

Die Kassette trägt wesentlich dazu bei, den Anfängern bei Programmfehlern zu helfen. Wenn Sie einen Fehler in Ihrem Programm-Modul machen, gibt der Computer entweder die Meldung aus, daß eine bestimmte Zeile falsch ist, oder er bringt ein unerwartetes Ergebnis. An diesem Punkt können Sie das Modul von der Kassette laden und es laufen lassen. Fehler sind so leichter zu finden.

Die Seite B der Kassette enthält drei vollständige Spiele:

1. PLNT32 oder PLNT16
2. AUTO1 oder AUTO2
3. MALPRG

Die ersten beiden sind Geschicklichkeitsspiele. Sie werden lernen, wie man sie entwirft und programmiert. Mit dem dritten, MALPRG, können Sie mit dem MSX-Computer farbige Bilder auf Ihrem Bildschirm erstellen. Damit der Einstieg erleichtert wird, sollten Sie zuerst diese Programme laufen lassen:

1. Schließen Sie den MSX-Computer an Ihr Fernsehgerät und an Ihren Kassetten-Recorder an.
2. Schalten Sie alle Geräte ein.
3. Wenn am Bildschirm die Meldung Ok erscheint, geben Sie:

   ```
   CLOAD (oder LOAD"CAS: bei AUTO2)
   ```

   ein und drücken Sie RETURN.
4. Legen Sie die Kassette in Ihren Kassetten-Recorder ein und drücken Sie PLAY.

Der Computer lädt das erste Programm, das er findet, von der Kassette in seinen Speicher. Wenn wieder Ok am Bildschirm erscheint, geben Sie RUN ein und drücken Sie RETURN. Jetzt läuft das Programm. (Kapitel 6 zeigt, wie man Programme abspeichert und wieder lädt.) Kapitel 14 erklärt, wie MALPRG verwendet wird.

Der Leser sollte sich nicht von dem etwas gewöhnungsbedürftigen Computer-Jargon entmutigen lassen. Sehen Sie sich die Ausdrücke im Begriffslexikon an. Denken Sie daran: erst Übung macht den Meister!

1

Einführung



1 Einführung

So wie fast jeder inzwischen ein Telefon hat, wird auch jeder bald einen Computer besitzen. Das bedeutet, daß die Programmierung und die Handhabung dieser Maschinen bald den Einzug in das tägliche Leben finden wird. Die Computer-Hersteller bieten ihre Produkte als nicht mehr wegzudenkende Gegenstände des täglichen Lebens an. Der Vergleich zwischen dem Telefon und dem Computer hinkt, wenn man die wachsende Bedeutung des Computers betrachtet. Es gibt nämlich einen großen Unterschied zwischen beiden. Normalerweise dient das Telefon nur der direkten mündlichen Kommunikation. Sonst kann man kaum etwas anderes damit machen. Im Gegensatz dazu kann der Computer innerhalb der Grenzen seiner Fähigkeiten für jede Aufgabe programmiert werden, wenn der Anwender die richtigen Befehle einsetzt. In diesem Sinne ist die Bedeutung des Computers als schnelle, wirkungsvolle und leistungsfähige Problemlösemaschine zu sehen. Der Schlüssel zu den Wundern der Computerwelt ist die Fertigkeit des Programmierens.

Ein erster Schritt in diese Richtung ist der Erwerb eines Computers. Ihr Ziel sollte es sein, die Eigenschaften Ihres Computers so gut wie möglich kennenzulernen.

Die Vermutung liegt nahe, daß bei dem großen Angebot an fertigen Programmen keine Notwendigkeit besteht, eine Programmiersprache zu erlernen. Bedenken Sie aber, daß die meisten Programme nicht für Heimcomputer, sondern für spezielle Zwecke entworfen wurden. Programmieren zu können heißt, Programmpakete Ihren Bedürfnissen entsprechend ändern zu können. Sicher werden Sie auch sehr schnell feststellen, daß es sehr befriedigend sein kann, Ihre eigenen Programme zu schreiben und den Computer dazu zu bringen, Aufgaben zu erledigen, die Sie ihm gestellt haben. Das Ziel dieses Buchs ist es, Ihnen die Grundlagen der Programmierung von MSX-Systemen zu vermitteln.

Erst mit einem Programm kann der Computer zum Leben erweckt werden. Die Beziehung Programm - Computer ist ähnlich der zwischen dem menschlichen Geist und dem menschlichen Körper. Es gibt keine Geheimnisse in Computer-Programmen. Vielleicht kann der Telefonvergleich helfen, den Prozess der Programmierung zu erläutern: Um einen bestimmten Gesprächsteilnehmer zu erreichen (= ein Computer-Programm zu schreiben), verwenden Sie ausschließlich Zahlen, (= Code oder reservierte Wörter), die sie in einer bestimmten Reihenfolge eingeben (= eine spezielle Grammatik oder Syntax). Man braucht wohl nicht darauf hinzuweisen, daß die strikte Einhaltung der Regeln in beiden Fällen notwendig ist, um das gewünschte Ergebnis zu erhalten. Ein Programm ist eine Ansammlung von codierten Anweisungen (reservierten Wörtern) mit einer besonderen, unbedingt einzuhaltenden Grammatik (Syntax), die der Computer versteht.

Ein Satz reservierter Wörter und die Syntaxvereinbarungen bilden eine Programmiersprache. Sie werden in diesem Buch die Programmiersprache MSX-BASIC kennenler-

nen. BASIC heißt: Beginners all Purpose Symbolic Instruction Code. Es enthält ungefähr 150 englische Wörter, die reservierte Wörter oder Schlüsselwörter genannt werden. BASIC ist besonders für all jene geeignet, die wenig oder keine Programmiererfahrung haben. Seine Syntax ist der englischen Sprache verwandt, so daß man sich verhältnismäßig leicht die Wörter merken und sie anwenden kann. Wie bei der menschlichen Sprache, gibt es auch hier Dialekte. Dieses Buch behandelt einen BASIC-Dialekt, der unter dem Namen MSX-BASIC bekannt ist. MSX heißt: Microsoft Software Exchange. Der Grund für die Wahl des MSX-BASIC wird in Kapitel 3 klar. Die Sprache ist eine erweiterte Version des Microsoft-Standard-BASIC 4.5. Sie unterstützt Grafik, Musik und Peripheriegeräte, die an den MSX-Heimcomputer angeschlossen werden können. Weitere Vorteile des MSX-BASIC lernen Sie kennen, wenn Sie dieses Buch durcharbeiten.

Da es für das Selbststudium geschrieben wurde, haben wir versucht, den langweiligen Stil technischer Handbücher zu vermeiden, so daß Sie mit Spaß und Freude lernen können und diese Begeisterung bis zum Ende dieses Buches anhält. Weil kaum ein Heimcomputer-Besitzer Videospielen abgeneigt ist, entwickeln wir ein völlig neues Spiel. Der Begriff 'spielend lernen' ist im Zusammenhang mit diesem Buch sicher nicht falsch gewählt. Wir werden jeden Schritt beim Programmieren des Videospiels in MSX-BASIC erklären. Mit jedem weiteren Programmodul lernen Sie die Bedeutung und den Gebrauch bestimmter Teile des MSX-BASIC-Vokabulars kennen. Nach der Fertigstellung des Videospiels (Kapitel 12) werden Sie die Grafik-, Sound- und Joystick-Steuer-Befehle meisterhaft beherrschen und Sie haben, ganz nebenbei, das strukturierte, modulare Programmieren in MSX gelernt.

1.1 Die Computer-Hardware

Sie müssen die Funktionen der verschiedenen Teile Ihres Computers kennen, damit Sie ein Programm-System laufenlassen können. Wir werden als Analogon eine Stereoanlage heranziehen. Möglicherweise wissen Sie nicht, wie eine Stereoanlage arbeitet. Die einzelnen Bestandteile werden Sie sicher kennen.

Der Verstärker allein kann keinen Ton erzeugen. Sie müssen ihn mit der Quelle der Musik, einem Signal vom Radio-Tuner oder einem Kassetten-Recorder verbinden. Außerdem brauchen Sie als Ausgabegerät einen Lautsprecher.

Wie der Verstärker bringt auch der Computer ohne eine Datenquelle (Tastatur, Kassetten-Recorder oder Floppy-Disk-Laufwerk) nichts zustande. Außerdem braucht er Ausgabegeräte wie einen Monitor oder einen Drucker. Im folgenden finden Sie die Analogie zur Stereoanlage grafisch dargestellt:

Stereo		**Computer**
	EINGABEGERÄT	
Radio-Tuner Kassettendeck		Tastatur Kassetten-Recorder Floppy-Disk-Laufwerk
	STEUERGRÄT	
Verstärker		Zentraleinheit (CPU)
	AUSGABEGERÄT	
Lautsprecher		Monitor Drucker
	SPEICHERGERÄT	
Kassettendeck		Kassetten-Recorder Floppy-Disk-Laufwerk

Die Teile Ihrer Ausrüstung, die das Computer-System bilden, nennt man Hardware. Nehmen Sie sich Zeit, sie möglichst gut kennenzulernen. Sie werden damit häufig konfrontiert werden.

Ähnlich wie eine Stereo-Anlage, die abgespeicherte Informationen braucht, um Töne zu erzeugen, arbeitet der Computer erst, wenn man ihm Anweisungen (das Programm) gibt. Erst dann weiß er, was er tun soll. Programme, die sich im Computer befinden, und Programme, die Sie dem Computer geben, werden Software genannt.

Damit Sie die in diesem Buch gezeigten Programme ausprobieren können, brauchen Sie folgende Hardware:

1. Einen MSX-Computer
2. Einen Kassetten-Recorder
3. Ein Fernsehgerät oder einen Monitor

Dazu gehören natürlich auch die entsprechenden Kabel, die die Geräte miteinander verbinden. Ein Drucker kann ebenfalls sinnvoll sein, damit Sie Ihre Programme besser lesen und Fehler leichter finden können.

Zusätzliche Software benötigen Sie nicht, da das MSX-BASIC bereits im MSX-Computer vorhanden ist.



2
Allgemeine Information

2 Allgemeine Informationen

2.1 Ein Überblick über Programmiersprachen

Computer führen ihre Funktionen mit Tausenden von Ein- und Ausschaltvorgängen durch. Mathematisch wird dies mit binären Zahlen dargestellt (Anhang C). In ihr kommen nur Einsen und Nullen vor. Diese Darstellungsart ist nicht sehr übersichtlich. Damit die Verständigung zwischen Mensch und Maschine leichter wird, wurden die Programmiersprachen entwickelt. Eine Programmiersprache besteht aus einer Reihe von Regeln, die die Kommunikation zwisch+en Programmierer und Computer erleichtern soll. Wie bei der menschlichen Sprache, gibt es auch hier verschiedene Sprachen und Dialekte, die man in drei Kategorien einordnen kann:

1. Maschinensprache
2. Assembler-Sprache
3. Höhere Programmiersprachen

MSX-BASIC ist eine höhere Programmiersprache. Eine Hochsprache bedeutet mehr Arbeit für den Computer, weil er die Hochsprachen-Befehle erst in die Maschinensprache umsetzen muß. Das bedeutet aber nicht, daß eine Hochsprache schwierig anzuwenden ist. Das Gegenteil ist der Fall!

Es ist sinnvoll, die Bedeutung von Speicherzuweisungen, Adressen, Ein-/Ausgabeports, Video-Display-Prozessoren und programmierbaren Soundgeneratoren zu kennen. Außerdem sollte man wissen, wie Zahlen dargestellt werden. Um Sie nicht zu sehr zu beanspruchen, werden wir die Diskussion dieser Themen auf ein Minimum beschränken.

2.2 Die Vorteile einer höheren Programmiersprache

Die höheren Programmiersprachen sind das Ergebnis der Forderung nach einer naturähnlichen Sprache, die die Programmier-Ideen einfacher umsetzbar machen sollte. Sie haben normalerweise einen Wortschatz von etwa 100 reservierten Wörtern. Sie sehen, daß Sie nichts über Maschinencode und Hardwarekonzepte wissen müssen, um ein Programm in einer höheren Programmsprache schreiben zu können.

Ein weiterer, oft aufgeführter Vorteil von Programmen, die in einer Hochsprache geschrieben sind, ist die Unabhängigkeit vom Computersystem. Programme können von einem Computer auf einen anderen übertragen werden, wenn man die gleiche Sprache benutzt. Schlimmstenfalls werden dabei kleinere Modifikationen erforderlich. Diese Software-Kompatibilität ist wünschenswert, aber leider sehr selten. Die unterschiedlichen Systeme und Formate, die viele Hersteller einsetzen, um ihre Produkte vor dem unerlaubten Kopieren zu schützen, führen dazu, daß Programmübertragungen nicht ohne weiteres möglich sind.

Programme, die mit MSX-BASIC entwickelt werden, laufen auf allen MSX-Rechnern.

2.3 Interpreter und Compiler

Ein Mikroprozessor kann nur Maschinenanweisungen ausführen. Mit Befehlen einer Hochsprache kann er nicht unmittelbar arbeiten. Compiler und Interpreter sind Programme, die die Hochsprachenanweisungen in Maschinensprache umwandeln.

MSX-BASIC ist ein Interpreter.

Ein Interpreter übersetzt ein Programm Zeile für Zeile während der Programmausführung. Im MSX-BASIC analysiert der Interpreter jede Anweisung, überprüft sie auf Fehler und führt dann die gewünschten Befehle aus. Dieser interpretative Prozeß wird jedesmal durchgeführt, wenn das Programm läuft.

Ein Compiler übersetzt ein Quellprogramm in ein Maschinenprogramm, das der Mikroprozessor direkt ausführen kann. Das Übersetzen findet nur einmal statt. Während des Programmlaufs wird nicht mehr übersetzt.

Um MSX-BASIC-Programme zu entwickeln, brauchen Sie keinen Compiler. Wir haben den Compiler nur erwähnt, damit Sie wissen wie er funktioniert und welchen Unterschied es zum Interpreter gibt.

2.4 Die Programmentwicklung

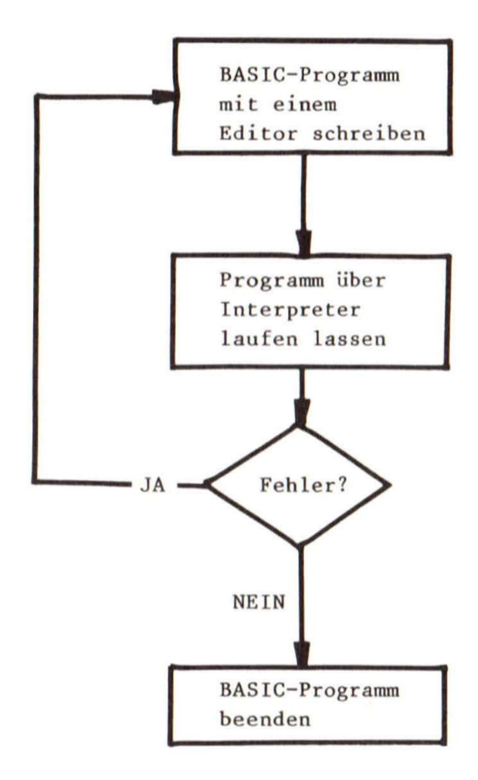
Ein Programm sollte in drei Schritten entwickelt werden:

1. Erstellen Sie eine BASIC-Quelldatei mit einem Editor-Programm. MSX-BASIC verfügt über einen eingebauten Bildschirm-Editor (Kapitel 5).

2. Machen Sie Ihr BASIC-Programm fehlerfrei. Sie können den MSX-BASIC-Interpreter benützen, um Syntax- und logische Fehler zu finden. Fehler werden mit einer Fehlermeldung angezeigt (Anhang B). Manchmal kann es vorkommen, daß unerwartete oder unerwünschte Ergebnisse die einzigen Fehler Ihres Programms sind. Verbessern Sie die Fehler und lassen Sie das Programm erneut laufen.

3. Ist Ihr Programm völlig fehlerfrei, können Sie beginnen, damit zu arbeiten.

Das Flußdiagramm zeigt diesen Ablauf:



3

Was ist MSX?



3 Was ist MSX?

3.1 Das MSX-Konzept

Neue Technologien werden in ihrem Anfangsstadium nur selten in das Korsett einer Norm gezwängt. In den frühen Tagen des Kassetten-Recorders gab es acht verschiedene Kassettensysteme. Heute ist die Übertragbarkeit von Kassetten kein Problem mehr. Sie können ohne weiteres eine Grundig-Kassette in einen Sony-Kassetten-Recorder einlegen und Musik hören. Da liegt es nahe, daß man die gleiche Kompatibilität bei Programmen erwartet. Leider ist man derzeit in der Rechnertechnik noch nicht so weit.

Software kann längst noch nicht beliebig zwischen Rechnern ausgetauscht werden. Einige Systeme lassen aber bereits den Austausch zwischen bestimmten Geräten zu (CP/M, PC-DOS, MS-DOS). Häufig gibt es selbst bei übereinstimmenden Programmausstattungen Schwierigkeiten, weil die unterschiedlichen Floppy-Disk-Formate, die von den verschiedenen Herstellern verwendet werden, nicht übereinstimmen. Eine Programmdiskette für den ABC-Computer kann nicht in einem XYZ-Computer eingesetzt werden, auch wenn beide eine 5-1/4-Inch-Floppy und das gleiche Betriebssystem benutzen. Es gibt so viele unterschiedliche Floppy-Disk-Formate auf dem Markt, daß sogar Experten gelegentlich verwirrt sind.

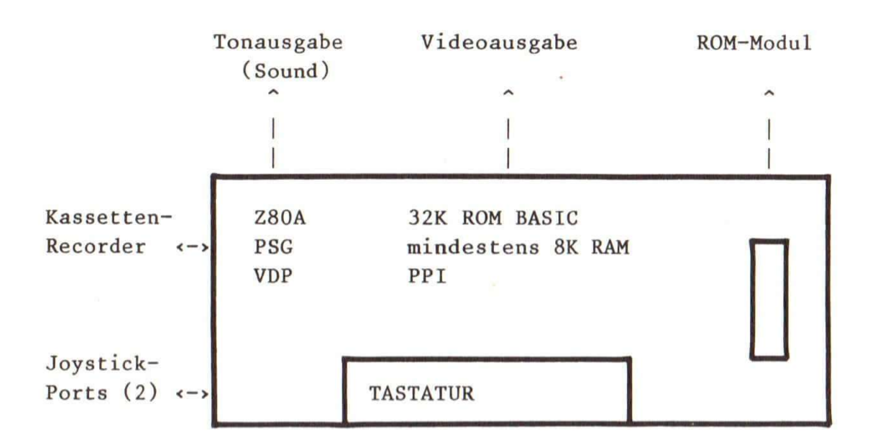
Der Einstieg der IBM in den Personalcomputer-Markt brachte einen Quasi-Standard für den Intel 8088-Mikroprozessor.

Auf dem Markt für preiswertere Heimcomputer herrscht weiterhin ein wildes Durcheinander.

Mit MSX (= Microsoft Software Exchange) wurde versucht, das Problem zu lösen, indem man die Sprache, die Anwendersoftware und die ROM-Module kompatibel machte. Die Idee stammt von der Microsoft Corporation, die MSX als Heimcomputer-Standard vorgeschlagen hat. Viele japanische Hersteller haben diesen Vorschlag sehr schnell übernommen. Inzwischen gibt es aber auch aus europäischer Fertigung interessante Geräte. Ein MSX-Heimcomputer-System muß folgenden Anforderungen entsprechen:

- Vollständige Software-Kompatibilität,
- Z80 CPU mit MSX-BASIC als Basissystem,
- Farbgrafik und Klangeffekte mit übereinstimmenden Merkmalen,
- einfach von einem Spielcomputer zu einem Heim- und Bussiness-Computer umkonfigurierbar.

Diese Anforderungen werden durch die Standardisierung der Hardware und der Software-Architektur erfüllt. Ein Minimalsystem besteht aus:



Ein MSX-Computer hat folgende Eigenschaften:

1. Er zeigt 16 Farben am Bildschirm an.
2. Er erzeugt gleichzeitig bis zu drei Töne.
3. Die Steuerung der Peripheriegeräte erfolgt über einen Einsteckschacht (ROM-Modul).
4. Es läuft eine BASIC-Sprache darauf.

Eine der Stärken des MSX-Konzepts sind die Einsteck-Module. Die Standardisierung geht hier soweit, daß alle Komponenten wie bei der Stereo-Anlage einfach miteinander verbunden werden können und zusammenarbeiten.

Software-Module, die mit MSX erzeugt worden sind, können ohne Änderung auf jedem beliebigen MSX-Computer laufen. Peripheriegeräte, die den MSX-Standard berücksichtigen, sind ohne Rücksicht auf deren Herkunft am Modulschacht anschließbar. Im nächsten Abschnitt beschäftigen wir uns mit diesem Konzept näher.

3.2 Der Einfluß der Rechnertechnik

Wir befinden uns zur Zeit in der Mitte der Umstrukturierung von der Industrie- zur 'Informierten Gesellschaft', in der der Heimcomputer eine wichtige Rolle spielt. Die fehlende Standardisierung von Software und Hardware hat bisher den weitverbreiteten Einsatz von Personalcomputern zu Hause verhindert.

Wenn kompatible Systeme in ausreichender Zahl auf den Markt kommen, wird die Produktion und Verteilung von billiger Software für Heimcomputer Wirklichkeit. Die Kompatibilität senkt die Software-Produktionskosten und verleiht dem Software-Markt Stabilität. Die Produkte erfreuen sich eines längeren Lebens, weil die Software besser wird. Möglicherweise wird mit besserer Software auch mehr Hardware verkauft, was dann dazu führt, daß noch bessere und attraktivere Software angeboten wird.

MSX ist ein preiswerter Heimcomputer-Standard. Es ist der einzige, der von 20 und mehr Konsumelektronik-Herstellern übernommen wurde. MSX ist für sie nicht nur ein EDV-technischer Standard unter vielen. Für sie ist es auch eine Chance, ihre anderen Produkte mit den MSX-Rechnern zusammenarbeiten zu lassen. Ein Beispiel dafür ist Yamaha. Dort wird der MSX-Rechner dazu verwendet, Musikinstrumente aus dem gleichen Hause zu steuern.

Folgender Trend ist abzusehen: MSX-Rechner werden gleichwertig neben Video-Recordern, Fernsehgeräten oder Videospielen ihren Platz einnehmen. Sie können ebenso als Alarmanlagen wie als Informations- und Unterhaltungszentren dienen.

Der Vorteil der Standardisierung wird, wie beim Kassetten-Recorder, bald zu erkennen sein.

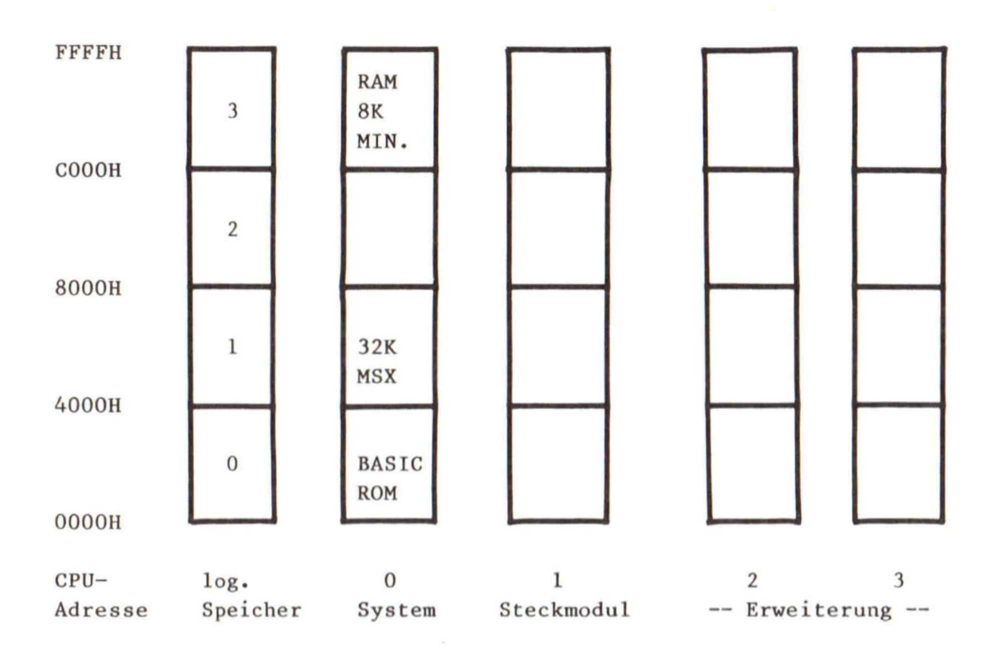
Wenn Sie gerade mit dem Programmieren anfangen, überspringen Sie besser den nächsten Abschnitt. In ihm besprechen wir die Vorteile des Steckplatz- und MSX-BASIC-Speicher-Konzepts. Wir werden dort auch auf das hexadezimale Zahlensystem eingehen. Wenn Sie Hilfestellung brauchen, sehen Sie in Anhang C nach.

3.3 Steckmodule und MSX-BASIC

In der MSX-Sprache hat der Ausdruck Steckplatz (engl.: SLOT) eine spezielle Bedeutung. Ein Slot erlaubt den Zugriff auf 64 KBytes Speicher (eingebaut oder anschließbar), die die CPU durch ein Slot-Anwahl-Signal adressieren kann. Das MSX-System be-

nötigt in seiner Minimalkonfiguration zwei Steckeinheiten: eine System- und eine ROM-Modul-Speichereinheit. Beide Slots müssen nicht die vollen 64 KByte RAM enthalten.

Im folgenden sehen Sie eine typische MSX-System-Speicheraufteilung:



Sie sehen hier das Minimalsystem mit vier Steckplätzen und zwei Einsteckeinheiten (BASIC SLOTS). Mit den sogenannten Erweiterungsmodulen (EXPANSION SLOTS) kann der ansprechbare Speicherbereich vergrößert werden.

Jede Einsteckeinheit mit je 64K Speicher ist in vier 16 KBytes-Blöcke unterteilt.

Die Systemeinheit (SLOT 0) besitzt in allen MSX-Systemen als Minimum 8K RAM. Das MSX-BASIC-Interpreter-ROM hat bis zu 32 KBytes mit den Adressen von 0000H bis 7FFFH. Der MSX-BASIC-Interpreter benutzt das größte verfügbare, zusammenhängende RAM in 16K-Blöcken. Die Adressen liegen zwischen 8000H und 0FFFFH. Diese RAM-Blöcke können in den verschiedenen SLOTS placiert sein. Mit dem SLOT-Anwahl-Register werden die entsprechenden Blöcke in den Steckeinheiten, die gerade angesprochen werden, adressiert. Theoretisch können die 64 KBytes Speicher von vier verschiedenen SLOTS kommen.

In einem System mit mehreren Speicher-Bänken erhält jedes Bauteil im gleichen Speichergebiet das gleiche Anwahl-Signal. Deshalb dürfen zwei oder mehr Bauteile nicht

die gleiche Speicheradresse haben, weil ein BUS-Konflikt das System zum Absturz bringen oder sogar Bauteile zerstören kann. Mit dem Steckplatz-Konzept können mehrere Bauteile die gleiche Adresse haben, müssen aber in verschiedenen SLOTS liegen. Auch Programme können an gleichen Startadressen beginnen, wenn sie sich in verschiedenen Steckeinheiten befinden.

Zusätzlich zur Systemeinheit hat jeder MSX-Computer einen ROM-Einschub (SLOT 1). Besitzt ein Gerät, das über dieses Steckmodul angeschlossen ist, einen neuen BASIC-Befehl, kann ihn der MSX-BASIC-Interpreter verwenden.

Zur Zeit können bis zu vier Steckmodule ohne Bus-Puffer angeschlossen werden. Damit man zusätzliche SLOTS bedienen kann, muß man einen Puffer einsetzen. Die CPU kann nicht entscheiden, ob ein Steckmodul vor oder nach dem Puffer vorhanden ist. Deshalb braucht man eine Schaltung, die anhand der Steckmodulsignale die Datenflußrichtung des Puffers festlegt.

Weitere diesbezügliche Erklärungen würden den Umfang des Buchs sprengen. Wir haben das Konzept hier nur vorgestellt, um zu zeigen, daß das SLOT-Konzept das System flexibler macht und vielfältige Erweiterungsmöglichkeiten bietet.



4

Die Haupteigenschaften des MSX-BASIC



4 Die Haupteigenschaften des MSX-BASIC

4.1 Wir beginnen, in MSX-BASIC zu programmieren

Das MSX-BASIC besitzt Ähnlichkeit mit dem GW-BASIC, dem Standard für 16-Bit-Systeme. Es ist eine erweiterte Version der Microsoft-Standard-BASIC-Version 4.5 mit Unterstützung für Grafik und Musik und für verschiedene Peripheriegeräte, die an den MSX-Heimcomputer angeschlossen werden können. Zahlen werden mit doppelter Genauigkeit (14 Stellen) dargestellt.

Das MSX-BASIC ist bereits im ROM vorhanden.

Ist Ihr Rechner unter Beachtung der Herstellervorschriften angeschlossen, wird das MSX-BASIC sofort nach dem Einschalten geladen. Am Bildschirm sollte folgendes zu sehen sein:

```
MSX BASIC VERSION 1.0
Copyright 1983 by Microsoft
xxxxx Bytes free
Ok
```

Ok bedeutet, daß BASIC bereit ist, Ihre Befehle anzunehmen.

xxxxx Bytes free zeigt die Größe Ihres verfügbaren RAMs an, auf das BASIC zugreifen kann. Der Wert hängt von der Ausrüstung Ihres Computers ab.

4.2 Betriebsarten

Erscheint MSX-BASIC mit der Meldung Ok am Bildschirm, befindet es sich in der Befehlsebene und ist bereit, Kommandos anzunehmen. Sie können BASIC entweder im direkten oder im indirekten Modus betreiben.

4.2.1 Direkter Modus

Im direkten Modus werden Anweisungen und Befehle nicht mit Zeilennummern versehen, sondern direkt ausgeführt, sobald die RETURN-Taste gedrückt wird. Ein Beispiel:

Der Schirm zeigt:	Sie tippen:
Ok	PRINT "HALLO" <RETURN>
HALLO	
Ok	

Das Ergebnis "HALLO" wird sofort angezeigt. In diesem Modus kann das Ergebnis von arithmetischen und logischen Operationen zum späteren Gebrauch gespeichert werden. Die Anweisungen selbst werden nicht abgelegt.

Variablen sind Einheiten, die durch ein Symbol oder einen Namen dargestellt werden (Abschnitt 4.3.2). Mit BASIC können Sie den Variablen im direkten Modus Werte zuweisen. Das Beispiel:

```
LET A = 1
LET B = 2
```

mit A und B als Variable übergibt den Wert 1 an A und den Wert 2 an B. Diese Zuweisungen sind gültig, solange Sie sich im direkten Modus befinden und können auch in nachfolgenden Ausdrücken verwendet werden. Geben Sie nun ein:

```
PRINT A+B
```

Der Wert 3 wird am Bildschirm angezeigt.

Lernen Sie gerade BASIC, dann eignet sich dieser Modus gut zum Austesten Ihrer Programm-Segmente. Er ist auch dann sinnvoll, wenn Sie BASIC als Taschenrechner für schnelle Berechnungen, die kein vollständiges Programm erfordern, einsetzen wollen.

4.2.2 Indirekter Modus

Der indirekte Modus wird für die Programmeingabe verwendet. Ein BASIC-Programm besteht aus Zeilen, die Anweisungen und Instruktionen enthalten. Jede Zeile beginnt mit einer Zeilennummer. Ein typisches Beispiel:

Zeilen-nummer	Anweisung reserv. Wort	Anweisung Text	Zeilen-abschluß
100	LET	A = 1	<RETURN>

```
        ^         ^
  Leerzeichen  Leerzeichen
```

Jede Programmzeile wird mit RETURN abgeschlossen.

Wenn der Befehl:

```
RUN <RETURN>
```

eingegeben wird, bekommen alle Variablen den Wert Null zugewiesen und das Programm wird im Speicher abgearbeitet.

Ein Beispiel eines BASIC-Programms:

```
10 LET A = 1
20 LET B = 2
30 PRINT A+B
```

Nachdem Sie RUN <RETURN> eingeben haben, zeigt der Bildschirm folgendes an:

```
3
Ok
```

Ein Programm können Sie sich als eine Folge von Befehlen, die vor jeder Anweisung eine Zeilennummer haben, vorstellen. Die Programmzeilen werden nicht sofort ausgeführt, sondern im Speicher abgelegt. Die Ausführung wird gestartet, wenn ein RUN-Befehl eingegeben wird. Das Programm wird immer sequentiell von der niedrigsten zur höchsten Zeilennummer durchlaufen. (Ausnahmen bilden direkte oder bedingte Sprunganweisungen.) Beachten Sie, daß die Zeilennumerierung von Ihnen abhängt, und daß Sie Abstände zwischen den einzelnen Zeilen lassen sollten, damit Sie später neue Zeilen einfügen können.

4.3 Der Zeichensatz und reservierte Wörter

4.3.1 Der Zeichensatz

Der MSX-BASIC-Zeichensatz besteht aus alphabetischen und numerischen, sowie grafischen und speziellen Zeichen. MSX-BASIC erkennt die Groß- und Kleinbuchstaben des Alphabets und die nu-merischen Ziffern 0 - 9. Außerdem die Sonderzeichen:

ZEICHEN	BEDEUTUNG
	Leerzeichen
;	Semikolon
=	Gleichheitszeichen
+	Plussymbol
-	Minussymbol
*	Sternchen
/	Schrägstrich
^	Exponentationssymbol
(	Klammer auf
)	Klammer zu
%	Prozentzeichen
#	Nummerzeichen
$	Dollarzeichen
!	Ausrufezeichen
[	eckige Klammer auf
]	eckige Klammer zu
,	Komma
.	Punkt
"	Anführungszeichen
'	Apostroph
:	Doppelpunkt
&	Kaufmanns-Und
?	Fragezeichen
<	kleiner als
>	größer als
\	Backslash
@	Klammeraffe
_	Unterstreichen
{	geschweifte Klammer auf
}	geschweifte Klammer zu
DEL	Löschzeichen
ESC	ESCAPE
TAB	bewegt den Cursor zur nächsten Tabulatorposition.
BS	Backspace (Rückwärtszeichen)
LINE FEED	bewegt den Cursor zur nächsten physikalischen Zeile.
RETURN	beendet die Eingabe einer Zeile.

4.3.2 Variable und reservierte Wörter

Variable sind Namen, denen man in BASIC-Programmen Werte zuweist. Vergleichen Sie Variable mit einer Schublade, in die man Werte hineinlegt, die man später wieder herausholen kann. Variable enthalten numerische Werte oder Zeichenketten mit alphabetischen Zeichen. MSX-BASIC erkennt folgenden Variablen-Typen:

TYP	TYP-ZUORDNUNGS-ZEICHEN	ANZAHL DER BYTES, DIE DIE WERTE DER ENTSPRECHENDEN VARIABLEN EINNEHMEN
Ganzzahl	%	2
einfache Genauigkeit	!	4
doppelte Genauigkeit	#	8
Zeichenkette (String)	$	3 Bytes + Inhalt der Zeichenkette

Mit dem Typ-Zuordnungszeichen, das am Ende der Variablen steht, wird der Typ bestimmt. Sie können auch DEFINT, DEFSGN, DEFDBL und DEFSTR verwenden, um Variablen-Typen zu definieren (Abschnitt 9.2). Denken Sie daran, daß in einem Zuweisungsausdruck die Variablen-Typen übereinstimmen müssen.

In MSX-BASIC können die Variablennamen beliebig lang sein, müssen aber mit einem Buchstaben beginnen. Nur zwei Zeichen werden beachtet. Variablennamen können aus Buchstaben und Zahlen bestehen. Beginnt eine Variable mit FN, wird sie als Aufruf zu einer anwenderdefinierten Funktion betrachtet. Ohne Typ-Zuordnung sind numerische Variablen doppelt genau. Ein Variablenname darf nicht aus einem reservierten Wort bestehen und auch nicht Teil davon sein.

Alle MSX-Befehle, Anweisungen, Funktionsnamen und Operatoren sind reservierte Wörter. Nachfolgend finden Sie eine Liste mit den MSX-BASIC-Schlüsselwörtern.

ABS	AND	ASC	ATN	ATTR$	AUTO	
BASE	BEEP	BLOAD	SAVE	BIN$		
CALL	CDBL	CHR$	CINT	CIRCLE	CLEAR	CLOSE
CLOAD	CLS	CMD	COLOR	CONT	COPY	COS
CSAVE	CSNG	CSRLIN	CVD	CVI	CVS	
DATA	DEF	DEFDBL	DEFINT	DEFSNG	DEFSTR	DELETE
DIM	DRAW	DSKF	DSKI$	DSKO$		

ELSE END EOF EQV ERASE ERL ERR
ERROR EXP

FIELD FILES FIX FN FOR FRE

GET GOSUB GOTO

HEX$

IF IMP INKEY$ INP INPUT INPUT$ INSTR
INT INTERVAL

KEY KILL

LEFT$ LEN LET LFILES LINE LIST LLIST
LOAD LOC LOCATE LOF LOG LPOS LPRINT

MAXFILES MERGE MID$ MKD$ MKI$ MKS$ MOD
MOTOR

NAME NEW NEXT NOT

OCT$ OFF ON OPEN OR OUT

PAD PAINT PDL PEEK PLAY POINT POKE
POS PRESET PRINT PSET PUT

READ REM RENUM RESTORE RESUME RETURN RIGHT$
RND RSET RUN

SAVE SCREEN SGN SIN SOUND SPACE$ SPC
SPRITE SPRITE$ SQR STEP STICK STEP STICK
STOP STR$ STRIG STRING$ SWAP

TAB TAN THEN TIME TO TROFF TRON
USING USR

VAL VARPTR VDP VPEEK VPOKE

WAIT WIDTH

XOR

Operatoren: + - * / ^ \' < = >

Weitere Informationen über die reservierten Wörter finden Sie in der MSX-BASIC-Befehlszusammenfassung in Anhang A.

4.4 Die Hauptmerkmale von MSX-BASIC

MSX-BASIC hat eine Menge Merkmale, die Ihnen die Programmerstellung erleichtern:

1. Vier Variablen-Typen: Integer (+32767), Fließkommadarstellung mit einfacher Genauigkeit (7 Stellen), Fließkommadarstellung mit doppelter Genauigkeit (16 Stellen) und Zeichenketten,

2. Möglichkeiten zur Fehlersuche (TRON/TROFF),

3. Softkeys, Joysticks, und Tabletts,

4. PEEK und POKE, mit denen Sie jede Speicherstelle lesen und beschreiben können,

5. VPEEK und VPOKE, mit denen Sie jede VRAM-Speicherstelle (Video-RAM) lesen und schreiben können,

6. SOUND, mit dem Sie direkt die Register des PSG (programmable sound generator = programmierbarer Tongenerator) steuern können,

7. AUTO und RENUM, mit denen Sie automatisch Zeilen eingeben und neu numerieren können,

8. Matrizen bis zu 255 Dimensionen (die maximale Zahl der Elemente hängt vom verfügbaren Speicher ab),

9. bool'sche Operatoren OR, AND, NOT, XOR, EQV und IMP,

10. formatierter Ausdruck (PRINT USING),

11. direkter Zugriff auf 255 I/O-Ports mit INP und OUT,

12. automatischer, vollbildschirmorientierter Editor für leichte Programmeingabe,

13. Aufruf von Assembler-Unterprogrammen,

14. verschachtelte IF...THEN...ELSE-Befehle,
15. ON Intervall GOSUB und ON SPRITE GOSUB,

16. einfache Handhabung der Grafik: LINE, CIRCLE, DRAW und PUT SPRITE-Anweisungen,

17. einfache Handhabung der Musiktechnik (PLAY),

18. zehn programmierbare Funktionstasten (Softtasten),

19. unbegrenzte Erweiterungen über die Steckmodultechnik.

Vier Eigenschaften des MSX-BASIC sind besonders hervorzuheben:

1. Der vollbildschirmorientierte Editor ist auf der Befehlsebene für die Programmeingabe immer vorhanden. Sie können den Cursor frei über den Bildschirm bewegen und damit Ihre Programmzeilen editieren (Kapitel 5).

2. Der PSG kann mit den MSX-Befehlen auf einfache Weise angesprochen werden. Er arbeitet unabhängig von der CPU, so daß er Musik machen kann, während der Computer etwas anderes tut.

3. Mit MSX-BASIC-Befehlen können mühelos Grafiken für Spiele und geschäftliche Anwendungen erzeugt werden.

4. Durch den Einsatz von Softkeys (=programmierbare Funktionstasten) können anwenderfreundliche Programme erstellt werden.

Diese Möglichkeiten machen MSX-BASIC sehr flexibel. Trotzdem ist der größte Vorteil die Erweiterbarkeit. Jedes Gerät, das über den Steckschacht angeschlossen wird, kann zusätzliche BASIC-Befehle, -Anweisungen oder -Funktionen enthalten, die das MSX-BASIC ebenfalls verwenden kann. So stehen beinahe unbegrenzte Erweiterungsmöglichkeiten des MSX-Computers zur Verfügung.

4.5 BASIC-Wortschatz

Anschließend finden Sie eine Liste der BASIC-Wörter, die in diesem Kapitel behandelt wurden. Die Zahl neben den Wörtern bezieht sich auf den Abschnitt, in dem sie besprochen worden sind

```
PRINT 4.2.1
LET   4.2.1
RUN   4.2.2
```

5
Programmieren in MSX-BASIC



5 Programmieren in MSX-BASIC

Programmieren bedeutet, einem Computer Befehle zu geben, damit er eine Aufgabe durchführt. Diese Aufgabe kann ganz einfach aber auch sehr kompliziert sein. Sie können zum Beispiel nur Ihren Namen ausdrucken oder ein Video-Spiel programmieren.

Nachdem das Gerät eingeschaltet wurde, meldet sich MSX-BASIC mit der Bereitschaftsanzeige Ok und wartet auf Eingaben von der Tastatur. In diesem Kapitel werden wir die Eingabe von BASIC-Programmen im indirekten Modus beschreiben. Diese Einführung bringt Sie Schritt für Schritt zu dem Punkt, an dem Sie BASIC-Programme schreiben können. Wir erheben nicht den Anspruch, alle vorhandenen Befehle zu beschreiben, wir geben Ihnen lediglich die erste Hilfestellung.

5.1 Vereinbarungen

Wir treffen folgende Vereinbarungen, um die Eingabe von Programmbeispielen und die Reaktion darauf in diesem Buch darzustellen:

[ ] Für optionale Eingaben werden eckige Klammern verwendet.

‹ › Eingaben, die der Anwender machen muß, werden in spitzen Klammern eingeschlossen. Steht innerhalb der spitzen Klammern Text in Großbuchstaben, muß der Anwender die angegebene Taste drücken. (Beispiel: ‹CTRL›) Ist der Text in Kleinbuchstaben geschrieben, muß der Anwender die entsprechende Angabe eintippen. Ein Beispiel: ‹Dateiname›.

{ } Die geschweiften Klammern zeigen an, daß der Anwender eine Auswahl aus einer Reihe angegebener Möglichkeiten treffen kann. Steht der Text nicht in eckigen Klammern, muß mindestens eine Eingabe gemacht werden.

| Sind die Eingaben durch einen Strich getrennt, kann entweder der eine oder der andere Ausdruck eingesetzt werden.

Alle Kommas, Doppelpunkte, Schrägstriche und Gleichheitszeichen müssen so eingegeben werden, wie sie zu sehen sind.

Ein "+" zeigt an, daß zwei Tasten gleichzeitig gedrückt werden müssen. So bedeutet ‹CTRL›+‹STOP›: Halten Sie die CTRL-Taste fest und drücken Sie die STOP-Taste.

5.2 Das Zeilenformat eines Programms

Ein BASIC-Programm besteht aus einer Sammlung von Programmzeilen:

```
nnnnn BASIC-Anweisung [: BASIC-Anweisung...] [Kommentar] <RETURN>
```

nnnnn zeigt die Zeilennummer an. Mit ihnen wird Reihenfolge bestimmt, in der die Programmzeilen abgespeichert werden. Zeilennummern werden als Referenzpunkt für Abzweigungen und beim Editieren gebraucht. Im MSX-BASIC müssen die Zeilennummern im Bereich von 0 bis 65529 liegen.

Jeder Zeilennummer folgt eine BASIC-Anweisung, die einen Ausdruck oder eine Anweisung enthält. Innerhalb einer Zeile können auch mehrere BASIC-Anweisungen stehen. Sie müssen durch einen Doppelpunkt (:) getrennt werden. Bemerkungen am Ende einer Zeile müssen nur durch einen Apostrophen abgesetzt werden. Die Gesamtanzahl der Zeichen in einer BASIC-Programmzeile darf 255 nicht übersteigen.

Jede Programmzeile muß mit einem <RETURN> abgeschlossen sein. Von nun an werden wir <RETURN> am Ende einer BASIC-Programmzeile nicht mehr angeben. Wir erwarten, daß Sie diese Taste von sich aus drücken.

Geben Sie das folgende BASIC-Programm ein, das Ihren Namen auf den Bildschirm bringt:

```
10 PRINT "Geben Sie Ihren Namen ein";     'Name?
20 INPUT A$                               'Tastatureingabe
30 PRINT "Hallo ";A$                      'Schirmausgabe
40 END                                    'Programmende
```

Starten Sie dieses Programm mit RUN:

AUSGABE AM BILDSCHIRM:	SIE GEBEN EIN:
	RUN <RETURN>
Geben Sie Ihren Namen ein?	DIETER <RETURN>
Hallo DIETER	
Ok	

Ausführungsanweisungen, wie PRINT "Hallo";A$, bringen BASIC dazu, etwas zu tun. Nicht ausführbare Anweisungen, wie zum Beispiel eine Bemerkung, die nach einem Apostroph steht, haben für den Programmlauf keine Bedeutung. Man setzt nicht ausführbare Befehle ein, um ein Programm lesbarer zu machen. Sie erklären, was in der

Zeile geschieht. Wir schreiben in unsere Programme eine Menge Kommentare, um die Funktionsweise zu erklären. Spricht ein ganzer Zeilenblock eine Funktion an, ist es besser, den Kommentar nur an den Anfang des Blocks zu stellen.

5.3 Die Programmeingabe

Nun haben Sie Ihr erstes BASIC-Programm geschrieben. Betrachten wir jetzt den bildschirmorientieren Editor des MSX-BASIC.

Der Editor übernimmt jedesmal nach der Ok-Bereitschaftsanzeige und vor einem RUN-Befehl die Kontrolle. Mit ihm können Sie am Bildschirm editieren.

Vertippen Sie sich bei der Eingabe, gehen Sie mit dem Cursor mit Hilfe der Pfeiltasten an die Stelle, an der Sie Ihren Fehler gemacht haben und überschreiben ihn.

Jede Zeile, die mit einer Zeilennummer beginnt, wird als Programmzeile betrachtet und folgendermaßen behandelt:

1. Ist die Zeilennummer in Ordnung, wird sie zum Programm im Speicher hinzugefügt.

2. Gibt es die Zeilennummer schon im Speicher, ersetzt die neue Zeile die schon existierende.

Eine logische Programmzeile (maximal 255 Zeichen) kann größer sein, als eine physikalische Zeile (maximal 40 Zeichen) am Bildschirm. Geben Sie am Schluß ‹RETURN› ein, wird die logische Zeile insgesamt als eine Programmzeile für BASIC interpretiert.

5.4 Wie man ein Programm ändert

Um ein bestehendes Programm zu ändern, müssen Sie:

1. den Bereich der Programmzeilen, den Sie editieren wollen auflisten,

2. den Cursor in der entsprechenden Zeile positionieren,

3. Ihre Änderungen eingeben, und

4. ‹RETURN› drücken, wobei der Cursor an einer beliebigen Stelle in der geänderten Zeile stehen kann. Eine Programmzeile ist innerhalb des BASIC-Programms erst dann geändert, wenn ‹RETURN› gedrückt wird.

5.4.1 Der LIST-Befehl

Mit dem LIST-Befehl kann das gesamte Programm oder ein Teil davon am Bildschirm angezeigt werden. Dazu muß das Programm im Speicher stehen. Sie können mit LIST nach der Ok-Bereitschaftsanzeige folgendermaßen arbeiten:

SIE GEBEN EIN:	REAKTION:
LIST ‹RETURN›	Das gesamte Programm, beginnend von der niedrigsten Zeilennummer an, wird ausgegeben. Der LIST-Vorgang wird entweder durch das Ende des Programms oder durch ‹CTRL›+‹STOP› beendet.
LIST ‹Zeile #› ‹RETURN›	Nur die angegebene Zeile wird angezeigt.
LIST‹Zeile#›-‹RETURN›	Alle numerierten Zeilen ab der angegebenen Zeile bis zum Programmende werden angezeigt.
LIST-‹Zeile #› ‹RETURN›	Alle Zeilen vom Anfang des Programms bis zur angegebenen Zeilennummer werden angezeigt.
LIST‹Zeile#1›-‹Zeile #2› ‹RETURN›	Die Zeilen von der ersten bis zur zweiten angegebenen Zeilennummer werden angezeigt.
LIST. ‹RETURN›	Die Zeile, an der sich gerade der Cursor befindet, wird angezeigt.

BASIC kehrt nach einem LIST in den Befehlsmodus zurück.

Wird das Programm am Bildschirm sehr schnell angezeigt, kann es mit <STOP> unterbrochen und mit <STOP> wieder neu gestartet werden. Mit <CTRL>+<STOP> wird die LIST-Anzeige beendet und in den BASIC-Befehlsmodus zurückgekehrt.

Hinweis:
Die Ausgabe kann anstelle des Bildschirms direkt auf den Drucker ausgegeben werden, wenn der Befehl LIST durch LLIST ersetzt wird.

5.4.2 Die Steuerzeichen des bildschirmorientierten Editors

Wird ein Bereich des Programms mit LIST angezeigt, kann der Text geändert werden. Der Editor verwendet besondere Steuerzeichen, damit der Cursor bewegt, Text eingefügt und Text gelöscht werden kann.

Die Tasten, mit denen der Cursor an eine Stelle des Bildschirms bewegt wird, sind:

SONDERTASTEN:	STEUERZEICHEN:	FUNKTION:
	<CTRL>+^ *	Cursor nach oben
	<CTRL>+_ *	Cursor nach unten
	<CTRL>+\ *	Cursor nach links
	<CTRL>+] *	Cursor nach rechts
<HOME>	<CTRL>+K *	Cursor in die linke obere Ecke des Schirms
<TAB>	<CTRL>+I	Cursor auf nächste TAB-Position
	<CTRL>+B *	Cursor rückwärts an den Anfang des vorhergehenden Wortes
	<CTRL>+F *	Cursor vorwärts an den Anfang nächsten Wortes
	<CTRL>+J *	Zeilenvorschub

Die Tasten, mit denen man Text einfügen kann, sind:

SONDERTASTEN:	STEUERZEICHEN:	FUNKTION:
<INS>	<CTRL>+R *	Einfügemodus ein-/ausschalten
	<CTRL>+N *	Cursor ans Ende der Zeile
<RETURN>	<CTRL>+M *	Nächste Zeile

Die Tasten, mit denen Text gelöscht werden kann, sind:

SONDERTASTEN:	STEUERZEICHEN:	FUNKTION:
<DEL>	<DEL>	Zeichen unter Cursor löschen
<BS>	<CTRL>+H	Rücktaste und Zeichen löschen
	<CTRL>+U *	Zeile löschen
	<CTRL>+E *	vom Cursor bis zum Ende der Zeile löschen
<CLS>	<CTRL>+L *	Bildschirm löschen
	<CTRL>+C	Änderungen in Zeile aufheben

Beachten Sie, daß MSX-BASIC nach dem Einschalten im Überschreibmodus (Nicht-Einfügemodus) erscheint, so daß das bestehende Zeichen unter dem Cursor durch ein neues Zeichen ersetzt wird. Drücken Sie die Taste <INS>, wird der Insert- oder Einfügemodus eingeschaltet. In diesem Modus nimmt der Cursor eine kleinere Gestalt an. Neu eingegebene Zeichen werden an der Stelle eingefügt, an der sich der Cursor befindet. Zeichen, die sich rechts vom Cursor befinden, werden dann nach rechts geschoben. Die Tasten oder Steuerzeichen, die mit einem Sternchen versehen sind, schalten den Insert-Modus wieder aus.

Die Änderungen einer Zeile werden erst nach dem Drücken von <RETURN> angenommen. Dabei spielt es keine Rolle, an welcher Stelle sich der Cursor in der Zeile befindet. Die Änderungen beziehen sich auf die gesamte logische Zeile, auch wenn sie aus mehreren physikalischen Zeilen besteht.

Wollen Sie in den MSX-BASIC-Befehlsmodus zurückkehren ohne die Änderungen einer Zeile tatsächlich auszuführen, müssen Sie <CTRL>+C drücken. In diesem Fall sehen Sie die Änderungen zwar am Bildschirm, sie werden aber nicht in den Arbeitsspeicher geschrieben. Um ganz sicherzugehen, sollten Sie sich Ihre Programmzeilen noch ein-

mal mit LIST auf den Schirm holen, damit Sie genau feststellen können, was im Speicher abgelegt ist.

Schreiben Sie ein paar Zeilen auf den Bildschirm. Üben Sie den Gebrauch des Cursors. Versuchen Sie Text zu löschen und einzufügen. Wenn Sie sich erst einmal mit dem Editor zurechtfinden, werden Sie es zu schätzen wissen, überall am Bildschirm editieren zu können.

5.5 Weitere Sondertasten

MSX-BASIC besitzt zehn programmierbare Funktionstasten (Softkeys), die in der letzten Zeile des Bildschirms angezeigt werden. Sie sollen Ihnen bei der Programmierung helfen. Wird eine der Tasten gedrückt, erscheint der zugewiesene Inhalt an der Cursor-Position. Nach dem Einschalten sind folgende Werte initialisiert:

TASTE	WERT
F1	COLOR l
F2	AUTO l
F3	GOTO l
F4	LIST l
F5	RUN <RETURN>
F6	COLOR 15,4,7 <RETURN>
F7	CLOAD"
F8	CONT <RETURN>
F9	LIST. <RETURN> oo
F10	<CLS> RUN <RETURN>

Der Buchstabe l bedeutet ein Leerzeichen und o heißt, daß der Cursor nach oben bewegt werden soll.

Für Anwender, die schon etwas besser durchblicken, können F1 bis F10 mit den KEY-Anweisungen neu definiert werden.

5.6 Wie man BASIC-Programme fehlerfrei zum Laufen bringt

Nachdem ein BASIC-Programm in den Speicher des Computers geschrieben wurde, kann es mit folgendem Befehl ausgeführt werden:

```
RUN <RETURN>
```

Die Ausführung beginnt mit der niedrigsten Zeilennummer und fährt dann mit der nächsthöheren fort. Dieser Ablauf wird nur unterbrochen, wenn das Programm auf eine Sprunganweisung trifft (Kapitel 11).

Führt MSX-BASIC ein Programm aus, kann es durch das Drücken von <STOP> unterbrochen werden. Ein weiteres <STOP> führt die Ausführung fort.

Das Drücken der Tasten <CTRL>+<STOP> beendet die Ausführung mit der Meldung:

```
Break in nnnnn
```

In diesem Fall ist nnnnn die Programmzeile, in der die Ausführung angehalten wird. BASIC kehrt zum Befehlsmodus zurück.

Kommen in einem BASIC-Programm Syntax-Fehler oder logische Fehler vor, läuft es nicht so ab, wie Sie es erwarten. Man sagt, das Programm enthält BUGS (engl. bugs = Wanzen).

Ein Syntax-Fehler erscheint, wenn Sie eine Programmzeile eingegeben haben, die die MSX-BASIC-Sprachregeln nicht berücksichtigt. Normalerweise sind das falsch geschriebene Schlüsselwörter oder falsch aufgebaute Programmzeilen. Wird ein Syntax-Fehler entdeckt, geht BASIC automatisch in den Editiermodus. Sie können die Zeile mit dem Fehler auflisten lassen und den Cursor mit dem Befehl

```
LIST. <RETURN>
```

an den Anfang der Zeile stellen. Im bildschirmorientierten Editor können Sie dann den Fehler korrigieren. Die interaktive Eigenschaft des BASIC-Interpreters macht es einfach, Fehler zu beseitigen.

Eine Reihe MSX-Fehlermeldungen scheinen nicht sehr sinnvoll. Was bedeutet zum Beispiel:

```
NEXT without FOR.
```

Lesen Sie die Fehlermeldung und schauen Sie sie im Anhang B nach. MSX versucht, Sie bei der Fehlerfindung zu unterstützen.

Es gibt auch logische Programmfehler: Die Ausführungsfolge läuft falsch ab oder es erscheint ein falsches Ergebnis. Der BASIC-Interpreter kann solche Fehler natürlich nicht entdecken. In den meisten Fällen befindet sich in Ihrem Programm ein Fehler. Es ist nicht sehr wahrscheinlich, daß der Fehler durch MSX-BASIC selbst verursacht wird.

Programme laufen selten auf Anhieb fehlerfrei. Die Vorstellung, Fehler zu suchen, sollte Sie nicht abschrecken. Die meisten Datensammlungen - Anträge, Reden, Bücher - haben mehrere Versionen. Deswegen ist es gar nicht so ungewöhnlich, daß ein Programm immer wieder überarbeitet wird. Sie sollten sich das Fehlersuchen als kreativen Prozeß vorstellen, der Ihre Ideen vertieft und sich damit abfinden, daß Sie immer wieder Fehler suchen müssen.

5.7 BASIC-Wortschatz

In diesem Kapitel wurden folgende BASIC-Wörter eingeführt:

INPUT	5.2
END	5.2
LIST	5.4.1
LLIST	5.4.1



6

Wie man Programme abspeichert und aneinander hängt



6 Wie man Programme abspeichert und aneinanderhängt

Es ist sehr mühsam, ein Programm immer wieder einzutippen. MSX-BASIC hat eine Menge Funktionen aufzuweisen, mit denen Sie Ihre Programme zum späteren Einsatz auf Kassette ablegen können. Nehmen Sie Ihr Hardware-Handbuch und verbinden Sie den Kassetten-Recorder mit dem Computer, bevor Sie mit dem Buch weiterarbeiten.

6.1 Dateien und Dateispezifikationen

Eine Datei ist eine Sammlung von zueinander in Beziehung stehenden Informationen und kann als eine Einheit betrachtet werden. Die Dateien, mit denen wir arbeiten, sind BASIC-Programme oder eine Sammlung von Daten, die in BASIC-Programmen verwendet werden.

Physikalische Dateien werden auf einer Kassette abgespeichert. Damit man sie wieder benutzen kann, muß man BASIC sagen, wo sich die Information befindet. Eine Datei wird durch die Dateispezifikation bestimmt:

‹Gerätename›: ‹Dateiname›

Der ‹Gerätename› sagt BASIC, welches Gerät zur Ein- oder Ausgabe dient. Der ‹Gerätename› besteht aus drei Buchstaben, denen ein Doppelpunkt folgt. Es werden folgende Geräte unterstützt:

GERÄTENAME	GERÄT
CAS:	Kassette
CRT:	Bildschirm
GRP:	Grafikbildschirm
LPT:	Drucker

Ein zusätzlicher ‹Gerätename› kann beispielsweise ein ROM-Modul sein.

Der ‹Dateiname› gibt BASIC an, nach welcher Datei es suchen soll. Der Name besteht aus einer Zeichenkette, die zwischen einem und sechs Zeichen lang sein darf. Benutzen Sie einen Namen, der mehr als sechs Zeichen hat, schneidet BASIC die zusätzli-

chen Zeichen ab, so daß nur sechs festgehalten werden. Folgende Namen werden von BASIC akzeptiert:

IHRE EINGABE:	BASIC ERKENNT:
MEINPROGRAMM	MEINPR
ALTPROGRAMM	ALTPRO
ABC	ABC

6.2 Wie man ein Programm abspeichert

Haben Sie ein BASIC-Programm geschrieben, können Sie es mit dem CSAVE-Befehl auf Kassette abspeichern. Die allgemeine Form lautet:

```
CSAVE"‹Dateiname›"[, ‹Baudrate›] ‹RETURN›
```

Der Dateiname muß die Regeln aus Kapitel 6.1 berücksichtigen.

‹Baudrate› ist ein Parameter (optional), der bestimmt, mit welcher Geschwindigkeit eine Kassette beschrieben oder gelesen werden soll. ‹Baudrate›=1 setzt die Geschwindigkeit auf 1200 Baud und ‹Baudrate›=2 steht für 2400 Baud.

Speichern wir ein Programm ab. Geben Sie ein:

```
10 PRINT "HALLO"
20 END
```

Wenn Sie das Programm laufen lassen, sehen sie am Bildschirm:

```
HALLO
Ok
```

Damit dieses Programm auf Kassette gespeichert wird, müssen Sie:

1. den Kassetten-Recorder auf Aufnahme stellen, und

2. den Befehl

```
CSAVE "MPROG"‹RETURN
```

eingeben.

3. Wird Ok angezeigt, ist das Programm abgespeichert.

Mit CSAVE wird die Datei in einem komprimierten Format abgespeichert. Das spart Platz. Es gibt allerdings Fälle, wo Dateien im ASCII-Format abgespeichert werden müssen. In diesem Fall müssen Sie den SAVE-Befehl, der in Abschnitt 6.5 vorgestellt wird, verwenden.

6.3 Wie man ein Programm lädt

Wollen Sie wieder programmieren? Möchten Sie ein vorher abgespeichertes Programm laden oder nachschauen, ob der vorhergehende SAVE-Befehl erfolgreich abgearbeitet wurde? Dazu brauchen Sie die Befehle NEW, CLOAD und CLOAD?

6.3.1 NEW

Mit NEW wird der Speicher gelöscht, der Fehlersuch-Modus ausgeschaltet und es werden alle Dateien geschlossen. Nach der Ausführung von NEW kehrt BASIC in den Befehls-Modus zurück. Sie sollten jedesmal vor der Eingabe eines neuen Programms NEW verwenden:

```
NEW <RETURN>
```

6.3.2 CLOAD

Die allgemeine Form des CLOAD-Befehls lautet:

```
CLOAD "<Dateiname>" <RETURN>
```

Mit CLOAD werden alle Dateien geschlossen und ein noch im Speicher stehendes Programm gelöscht. Wird <Dateiname> nicht angegeben, wird die nächste auf der Kassette stehende Datei geladen. So verwendet man CLOAD:

1. Spulen Sie das Kassetten-Band an eine Stelle vor dem Programm, das Sie laden wollen.

2. Stellen Sie beim Recorder den Wiedergabemodus ein (Taste PLAY).

3. Geben Sie ein:

```
CLOAD "<Dateiname>"<RETURN>
```

4. War der Ladevorgang erfolgreich, zeigt der Computer:

```
Found:Dateiname
Ok
```

Als Übung wollen wir "MPROG" von der Kassette laden. Als Befehl für Schritt 3 geben Sie ein:

```
CLOAD "MPROG"<RETURN>
```

Der Computer antwortet mit:

```
Found:MPROG
Ok
```

Sehen Sie sich das Programm mit LIST an oder lassen Sie es mit RUN laufen, damit Sie sehen, ob CLOAD erfolgreich war.

6.3.3 CLOAD?

Da Sie Ihre Programm wahrscheinlich immer wieder ändern werden, haben Sie irgendwann mehrere Versionen des gleichen Programms. BASIC stellt einen Befehl zur Verfügung, mit dem Sie das Programm auf der Kassette mit dem im Speicher vergleichen können. Der Befehl lautet:

```
CLOAD? ["<Dateiname>"]
```

Wird kein Dateiname angegeben, wird das nächste auf der Kassette vorkommende Programm mit dem im Speicher befindlichen verglichen.

Bei allen Kassetten-Leseoperationen wird die Übertragungsgeschwindigkeit automatisch eingestellt.

Die Speicher- und Ladeprogramm-Operationen sind sehr wichtig, da wir im Laufe des Buchs mehrmals damit arbeiten müssen.

6.4 Strukturiertes Programmieren

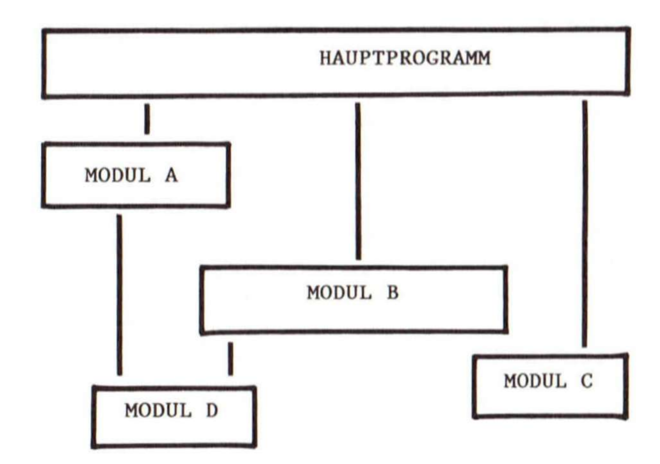
Komplexe Programme enthalten oft logische Fehler, die schwer zu finden sind. Um die Zahl der logischen Fehler zu reduzieren und die Fehlersuche zu erleichtern, wurde eine besondere Technik, die des strukturierten Programmierens, entwickelt.

Komplexe Programme enthalten häufig Bedingungen und davon abhängig Sprünge. Die BASIC-Sprunganweisungen lauten:

```
GOTO
GOSUB
IF...THEN...ELSE usw.
```

Werden diese Verzweigungsanweisungen allzu sorglos eingesetzt, können sie dafür sorgen, daß das Programm in einer Dauerschleife läuft oder Teile des Programms nicht durchgeführt werden. Strukturiertes Programmieren bedeutet, daß man eine Reihe einfacher Programm-Module, die Unterprogramme, schreibt, die als Bausteine für ein komplexes Programm dienen. Diese Technik nennt man modulares Programmieren.

Ein typisch strukturiertes Programm:



In diesem Fall ruft das Hauptprogramm die Module A,B und C auf, während die Module A und B das Modul D aufrufen.

In Verbindung mit dem modularen Programmieren setzen wir die TOP-DOWN-Entwurfsmethode ein. Zuerst schreiben wir das Hauptprogramm. Damit die Logik besser getestet werden kann, halten wir es sehr einfach und führen nur sehr wenige Berechnungen aus. Die meisten Berechnungen werden in den Unterprogramm-Modulen vorgenommen. Das Hauptprogramm kann mit Ersatzprogrammen getestet werden, damit man sieht, ob die Daten richtig übergeben werden. Anschließend wird jedes Modul ohne Hauptprogramm geprüft. Diese Technik unterteilt ein komplexes Programm in einfache Unterprogramme. Tritt ein Fehler auf, befindet er sich in einem kleinen Modul und kann leicht gefunden werden.

6.4.1 Die GOSUB...RETURN-Anweisung

Die MSX-BASIC-Anweisungen, mit denen man in ein Unterprogramm springen und von dort zurückkehren kann, lauten:

```
GOSUB ... RETURN.
```

Das allgemeine Format für GOSUB:

```
GOSUB <Zeilennummer>
```

<Zeilennummer> entspricht der ersten Zeile des Unterprogramms, auf die gesprungen werden soll.

Ein Unterprogramm kann beliebig oft von einem Programm und von einem anderen Unterprogramm aufgerufen werden. Das allgemeine Format für RETURN lautet:

```
RETURN <Zeilennummer>
```

Eine RETURN-Anweisung ohne <Zeilennummer> in einem Unterprogramm veranlaßt BASIC, an die Stelle des Programms zu springen, die nach dem letzten GOSUB-Befehl stand. Steht eine RETURN-Anweisung mit einer <Zeilennummer>, springt BASIC zu der Zeile, die durch <Zeilennummer> angesprochen wird.

Ein Unterprogramm kann mehr als ein RETURN enthalten, wenn die Programmlogik ein RETURN an verschiedenen Stellen des Unterprogramms erlaubt.

Betrachten Sie das folgende BASIC-Programm, damit Sie das Konzept verstehen:

```
10 GOSUB 1000                'BILDSCHIRM INITIIEREN
20 GOSUB 100                 'EINGABE
30 GOSUB 200                 'DAS HAUPTZEICHEN EINGEBEN
40 GOSUB 300                 'DEN GEGNER BEWEGEN
50 GOTO 20
100                          'DAS MODUL STUB EINGEBEN
110 PRINT "EINGABEMODUL"
120 RETURN
200                          'DAS HAUPTZEICHENMODUL STUB BEWEGEN
210 PRINT "HAUPTZEICHENMODUL"
220 RETURN
300                          'DAS FEINDMODUL STUB BEWEGEN
310 PRINT "FEINDMODUL BEWEGEN"
310 RETURN
1000                         'BILDSCHIRMMODUL STUB INITIIEREN
1010 PRINT "MODUL INITIIEREN"
1020 RETURN
```

Die Zeilen 10 bis 50 sind das Hauptprogramm. Sie rufen das Unterprogramm auf, das den Bildschirm initialisiert, fragen nach der Eingabe, bewegen das Hauptzeichen und den Gegner. Jedes aufgerufene Unterprogramm ist ein Fragment, das nur aus dem Hinweis besteht, daß es aufgerufen wurde. So kann die Logik des Hauptprogramms leicht getestet und jedes Unterprogramm-Modul unabhängig davon überprüft werden.

Diese Vorgangsweise hilft, Programme zu erstellen, die relativ fehlerfrei sind. Allerdings werden Fehler immer auftreten, egal wie geübt ein Programmierer ist. Ein Anfänger sollte sich nicht entmutigen lassen, denn das Fehlersuchen muß als ein Teil des Programmierens betrachtet werden.

6.5 Die Befehle SAVE und MERGE

Werden Unterprogramme als Einzelbestandteile eines Programms geschrieben, kann jedes Modul auch einzeln mit SAVE abgespeichert werden. Um das Gesamtprogramm zu erstellen, müssen sie mit MERGE aneinandergehängt werden. Die BASIC-Befehle, die diese Schritte unterstützen, sind SAVE und MERGE.

6.5.1 SAVE

Der SAVE-Befehl wird verwendet, um Programme abzuspeichern. Die allgemeine Form lautet:

```
SAVE "‹Gerät›:[‹Dateiname›]"‹RETURN›
```

‹Gerät›: bezeichnet ein Gerät, das das Programm aufnehmen soll (zum Beispiel CAS, CRT, LPT, oder GRP). In den meisten Fällen wird das der Kassetten-Recorder (CAS:) sein.

‹Dateiname›: ist eine Zeichenkette, die nach den MSX-BASIC-Regeln für Dateinamen gebildet werden muß.

Befindet sich das Programm:

```
10 PRINT "Hallo"
20 PRINT "Diese Zeile ersetzt die Zeile im Speicher."
```

im Speicher, bringt der Vorgang:

1. Schalten Sie den Kassetten-Recorder in den Aufnahme- Modus.

2. Geben Sie den Befehl ein:

```
SAVE "CAS:MPROG"
```

das Programm im ASCII-Format auf die Kassette. Wir beabsichtigen, das Modul an ein anderes mit MERGE anzuhängen.

6.5.2 MERGE

Mit dem Befehl MERGE wird ein Programm-Modul, das im ASCII-Format mit SAVE abgelegt wurde an das bestehende Programm, das im Speicher liegt, angehängt. Besitzen einige Zeilen der Datei auf Kassette die gleichen Zeilennummern wie das Programm im Speicher, ersetzen die Zeilen von der Kassette die im Speicher. Das allgemeine Format für MERGE lautet:

```
MERGE "‹Gerät›:[‹Dateiname›]"‹RETURN›
```

‹Gerät›: bezeichnet das Gerät, von dem das Programm-Modul geladen wird.

<Dateiname>: sagt BASIC, welche Datei angehängt werden soll. Wird der Name nicht angegeben, wird das nächste Programm, das sich auf der Kassette befindet, genommen.

Nehmen wir an, daß folgendes Programm im Speicher ist:

```
20 PRINT "Diese Zeile. wird ersetzt"
30 LET A = 1
40 LET B = 2
50 PRINT A+B
```

Damit das Programm "MPROG", das in Abschnitt 6.5.1 abgelegt wurde, an das darauffolgende angehängt wird, muß dieser Vorgang ablaufen:

1. Spulen Sie die Kassette zurück.

2. Drücken Sie die Play-Taste auf dem Kassetten-Recorder.

3. Geben Sie den Befehl:

```
MERGE "CAS:MPROG" <RETURN>
```

   ein.

Wenn die Ok-Bereitschaftsanzeige wieder erscheint, können Sie mit LIST das Programm und die Wirkung von MERGE anschauen. Es sollte so aussehen:

```
10 PRINT "Hallo"
20 PRINT "Diese Zeile ersetzt die Zeile im Speicher"
30 LET A = 1
40 LET B = 2
50 PRINT A+B
```

Beachten Sie, daß die Zeile 20 von der Kassette die Zeile mit der gleichen Nummer im Speicher überschrieben hat. Nach jedem MERGE-Befehl kehrt BASIC in den Befehlsmodus zurück.

Wir werden die SAVE- und MERGE-Befehle verwenden, um Unterprogramme in komplexe Programme einzubinden und um zu zeigen, wie die hervorragende Grafik und der SOUND-Befehl des MSX-BASIC funktionieren. Wir werden Unterprogramm-Module erstellen und sie dann für Aktionsspiele zusammenhängen.

6.6 BASIC-Wortschatz

In diesem Kapitel haben wir die folgenden neuen MSX-Befehle kennengelernt:

CSAVE	6.2
NEW	6.3.1
CLOAD	6.3.2
CLOAD?	6.3.3
GOSUB	6.4.1
RETURN	6.4.1
SAV	6.5.1
MERGE	6.5.2

7

Der Bildschirm



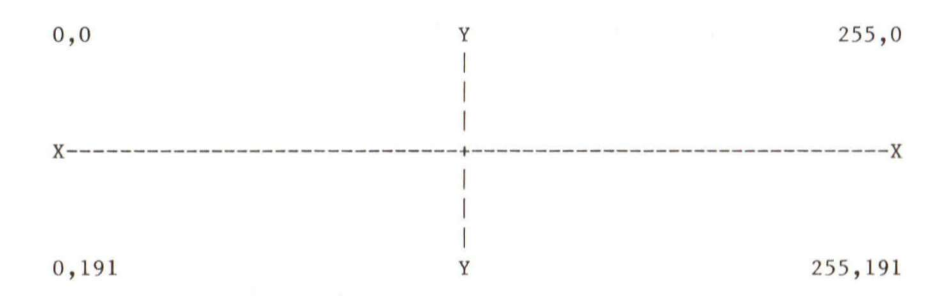
7 Der Bildschirm

Um ein Video-Spiel unter Verwendung von MSX-BASIC erstellen zu können, muß man das Bildschirmsystem verstehen.

Die Bildschirmauflösung entspricht der Dichte der einzelnen Bildpunkte (Pixels) am Bildschirm. Die Bildschirmauflösung von MSX beträgt 256 horizontal und 192 vertikal adressierbare Punkte.

MSX verwendet folgendes Koordinatensystem:

Die waagrechten Punkte werden der X-Achse zugewiesen und die senkrechten Punkte der Y-Achse. Punkt 0,0 ist der 1. Punkt in der linken oberen Ecke des Bildschirms, Punkt 255,0 ist an der rechten oberen Ecke, Punkt 0,191 ist an der unteren linken Ecke und Punkt 255,191 ist an der unteren rechten Ecke.



Diese Koordinatenvereinbarung verwenden wir im gesamten Buch. In den nachfolgenden Abschnitten präsentieren wir die folgenden MSX-BASIC-Anweisungen als Grundlage für weitere Grafikdarstellungen.

SCREEN	COLOR	
POINT	PSET	PRESET
LOCATE	POS	CSRLIN

7.1 Die SCREEN-Anweisung

Mit der SCREEN-Anweisung können Sie den SCREEN-Modus (Bildschirmmodus) zuweisen, die Sprite-Größe und den KEY-Klick setzen, die Kassetten-Baudrate einstellen und den Drucker anwählen. Das allgemeine Format lautet:

```
SCREEN [‹Bildschirm-Modus›][,‹Sprite-Größe›][,‹KEY-Klick›]
        [,‹Kassetten-Baudrate›][,‹Druckeroption›]
```

Im ‹SCREEN-Modus› sind die Werte 0 - 3 erlaubt, wobei:

0 40 Zeichen bei 24 Zeilen Textmodus anwählt,
1 32 Zeichen bei 24 Zeilen Textmodus anwählt,
2 den hochauflösenden Grafikmodus anwählt und
3 den niedrigauflösenden Grafikmodus anwählt.

Beachten Sie, daß aus dem SCREEN-Modus in den Textmodus zurückgekehrt wird, wenn eine INPUT-Anweisung eingegeben wird oder wenn BASIC in den Befehlsmodus zurückgeht. Im Textmodus lösen alle Grafikanweisungen (außer "PUT SPRITE", das eine legale Funktion im SCREEN-1-Modus ist) einen "Illegal Function Call"- Fehler aus.

Mit ‹Sprite-Größe› wird die Größe der Sprites bestimmt. Die Werte von 0 bis 3 haben folgende Bedeutung:

0 Wählt 8 x 8 Pixel unvergrößerte Sprites an.
1 Wählt 8 x 8 vergrößerte Sprites mit 4 Pixels pro Punkt an.
2 Wählt 16 x 16 Pixel unvergrößerte Sprites an.
3 Wählt 16 x16 vergrößertes Sprites mit 4 Pixels pro Punkt an.

‹KEY-Klick› schaltet den Tastenklick ein oder aus. Der Wert 0 schaltet ihn aus, während ein Wert ungleich 0 ihn einschaltet.

‹Kassetten-Baudrate› setzt die Baudrate für das Schreiben auf der Kassette. Die Werte 1 bis 2 sind verfügbar:

1 Wählt 1200 Baud.
2 Wählt 2400 Baud.

Beachten Sie, daß Sie die Baudrate zum Schreiben auf die Kassette auch beim CSAVE-Befehl mit der Baudraten-Option setzen können. Wird von der Kassette gelesen, wird die Baudrate automatisch bestimmt, so daß der Anwender nicht wissen muß, mit welcher Geschwindigkeit die Kassette beschrieben wurde.

Mit ‹Druckeroption› wird bestimmt, ob der Drucker ein MSX-Drucker mit Grafikfähigkeiten ist oder nicht. Ein Wert ungleich 0 repräsentiert einen MSX-Drucker, während 0 einen nicht MSX-Drucker anzeigt.

Bei Bewegungsspielen wird der hochauflösende SCREEN-Modus 2 am häufigsten verwendet.

```
SCREEN 2,0,0,1
```

wählt die hochauflösende Grafik an, setzt die Sprites auf unvergrößerte 8 x 8 Pixels, schaltet den Tastenklick aus und setzt eine Geschwindigkeit von 1200 Baud.

7.2 Die COLOR-Anweisung

Die MSX-Video-Platine unterstützt 16 Farben. Mit der COLOR-Anweisung wird der Vordergrund, der Hintergrund und die Randfarbe des Bildes am Schirm bestimmt. Das Format lautet:

```
COLOR [‹Vordergrund›][, ‹Hintergrund›][, ‹Rand›]
```

Die Argumente ‹Vordergrund›, ‹Hintergrund› und ‹Rand› können die Werte von 0 bis 15 annehmen. Für jeden Wert gelten folgende Farbzuweisungen:

0	Transparent
1	Schwarz
2	Grün
3	Hellgrün
4	Dunkelblau
5	Hellblau
6	Dunkelrot
7	Türkis
8	Rot
9	Hellrot
10	Dunkelgelb
11	Hellgelb
12	Dunkelgrün
13	Violett
14	Grau
15	Weiß

Der Standardwert ist Color 15,4,7:

Weißer Vordergrund, blauer Hintergrund und türkiser Bildschirmrand.

7.3 Wie man Pixels setzt

Nun kennen Sie sich mit den Bildschirm-Koordinaten und den Farbwerten aus und sind jetzt in der Lage, die Bildpunkte zu placieren und sie ein- oder auszuschalten. Die drei MSX-BASIC-Anweisungen, mit denen man die X- und Y-Pixel-Koordinaten setzt, sind POINT, PSET und PRESET.

Mit der POINT-Funktion liest man den Farbwert eines Pixels vom Bildschirm. Der Befehl

```
POINT (X,Y)
```

gibt eine Ganzzahl zwischen 0 und 15 zurück, die von dem Farbwert des Pixels an der Stelle X,Y abhängt. Ist der Punkt X,Y außerhalb des Bereichs, wird der Wert -1 zurückgegeben.

Mit dem PSET-Befehl wird ein Punkt an einer bestimmten Stelle des Bildschirms 'eingeschaltet'. Es gibt zwei Formate:

```
Format 1:   PSET (X-Koordinate, Y-Koordinate) [, <Farbe>]
Format 2:   PSET STEP (X-Offset, Y-Offset) [, <Farbe>]
```

Die Argumente in den Klammern sind die Koordinaten des Punktes, den Sie setzen wollen. Format 1 verwendet absolute Koordinaten, die einen Punkt mit den genauen Adreßangaben am Bildschirm festlegen. Ein Beispiel:

```
  PSET (8,6)

0,0 .........
            .
            .
            .      PSET setzt den Punkt an die durch den Stern
            .      gekennzeichnete Stelle.
            .
            * 8,6
```

Wollen Sie einen Punkt relativ zum letzten angegebenen Punkt setzen, müssen Sie das Format 2 verwenden. Nehmen wir das obige Beispiel, um den Punkt acht Stellen in waagrechter Richtung von (8,6) entfernt einzuschalten. Der Befehl lautet:

```
PSET STEP (8,0)    'Offset 8 in X und 0 in Y Richtung
```

Die absolute Adresse dieses Punktes ist dann (16,8).
Das letzte Argument von PSET bestimmt die Farbe, die der angewählte Punkt haben soll. Wird dieses Argument nicht geschrieben, wird die Vordergrundfarbe hergenommen. Sie können PSET auch verwenden, um Punkte auszuschalten. Wie? Indem Sie PSET mit einem Farbargument einsetzen, das der Hintergrundfarbe entspricht.

PRESET ist vom Format her zu PSET identisch:

```
Format 1:    PRESET (X-Koordinate, Y-Koordinate) [, ‹Farbe›]
Format 2:    PRESET STEP (X-Offset, Y-Offset) [, ‹Farbe›]
```

Der Farbwert von PRESET entspricht dem der Hintergrundfarbe und läßt deshalb alle Punkte verschwinden. Wird ein Farbargument angegeben, hat PRESET die gleiche Eigenschaft wie PSET.

Wird eine Koordinate gesetzt, die außerhalb des Bereichs liegt, findet weder bei PSET noch bei PRESET eine Aktion statt und es wird auch kein Fehler angegeben. Ist der Wert des Farbarguments größer als 15, wird eine "Illegal Function Call"-Fehlermeldung ausgegeben.

7.4 Die Cursor-Positionierung

Im MSX-BASIC-Textmodus (SCREEN 0,,,, und SCREEN 1,,,,) gibt es drei Cursor-Positionierungsanweisungen, die Reihen und Spalten als Argumente verwenden. Sie werden bei der Erstellung von Videospielen nicht verwendet. Wir werden sie hier aber der Vollständigkeit halber angeben. Die Anweisungen lauten:

```
LOCATE, CSRLIN und POS.
```

Das Format für LOCATE:

```
LOCATE [‹Reihe›],[‹Spalte›],[‹Cursor›]
```

Der Cursor wird zur Zeilennummer ‹Reihe› (1-24) und Spaltennummer ‹Spalte› (1-32 im SCREEN-1- oder 1-40 im SCREEN-0-Modus) bewegt. Das letzte Argument schaltet den Cursor ein (ungleich 0) oder aus (= 0).

Die CSRLIN-Funktion gibt die momentane Zeilenposition des Cursors zurück. Sie hat das Format:

```
Y = CSRLIN
```

Y ist eine numerische Variable, die den zurückgegebenen Wert (zwischen 1 und 24) enthält.

Die POS-Funktion übergibt die momentane Spaltenposition des Cursors. Sie hat das Format:

```
X = POS (0)
```

X liegt im Bereich von 1 bis 40 für SCREEN 0 und von 1 bis 32 für SCREEN 1.

7.5 Zusammenfassung und BASIC-Wortschatz

Wir haben gelernt, daß ein Pixel der kleinste Punkt ist, der am Bildschirm adressiert werden kann. Im hochauflösenden Grafik-Modus (SCREEN 2) können wir 256 x 162 Pixels am Bildschirm adressieren. Jedes Pixel kann dabei eine von 16 Farben enthalten. Folgende BASIC-Wörter haben wir in diesem Kapitel kennengelernt:

SCREEN	7.1
COLOR	7.2
POINT	7.3
PSET	7.3
PRESET	7.3
LOCATE	7.4
CSRLIN	7.4
POS	7.4

In den nächsten Kapiteln werden wir zusätzliche Farbgrafik-Anweisungen kennenlernen, mit denen wir Videospiele schreiben wollen.

8

Die Farbgrafik



8 Die Farbgrafik

Die grafischen Anweisungen, die im Grafik-Modus (SCREEN 2 oder SCREEN 3) verfügbar sind, und mit denen man im MSX-BASIC Bilder erstellen kann, lauten:

```
LINE   CIRCLE   PAINT   DRAW
```

Wir verwenden sie, um in diesem Kapitel eine einfache Straßenszene zu erstellen.

8.1 Die LINE-Anweisung

Die LINE-Anweisung zeichnet Linien am Bildschirm (absolut oder relativ). Man kann damit auch 'leere' und ausgefüllte Rechtecke erstellen. Die beiden Hauptformate lauten:

Format 1:

```
LINE [(X1,Y1)]-(X2,Y2)[, <Farbe>][, <B>:, <BF>]
```

Format 2:

```
LINE [STEP(X1,Y1)]-STEP(X2,Y2)[, <Farbe>][, <B>:, <BF>]
```

Die einfachste Form von LINE lautet:

```
LINE -(X2,Y2)
```

Damit wird eine Linie von der letzten Position nach (X2,Y2) in der Vordergrundfarbe gezogen. Wenn wir den Startpunkt angeben:

```
LINE (X1,Y1)-(X2,Y2)
```

wird eine Linie von (X1,Y1) nach (X2,Y2) gezogen. Wir können auch noch einen Farbcode dazuhängen:

```
LINE (X1,Y1)-(X2,Y2),4
```

Damit wird eine dunkelblaue Linie von (X1,Y1) nach (X2,Y2) gezogen. Das abschließende Argument für LINE ist ",B" für BOX = Rechteck oder ",BF" für FILLED BOX = gefülltes Rechteck. Mit ",B" wird MSX-BASIC gesagt, daß es ein Rechteck mit den gegenüberliegenden Ecken (X1,Y1) und (X2,Y2) zeichnen soll. Das Argument ",BF" zeichnet ein Rechteck und füllt das Innere mit der angewählten Farbe aus.

Zeichnen wir nun eine Straße und einige Häuser. Geben Sie das folgende Programm-Modul ein. Wenn Sie die Kassette mit den Beispielprogrammen aus dem Buch erworben haben, können Sie auch das Programm "SZENE1" von Seite A laden.

```
100 COLOR 15,9,1       'vordergrund,hintergrund,farbe setzen
110 SCREEN 2,3         'grafikmodus w/16x16 sprite setzen
245                    'haeuser zeichnen
250 LINE(0,50)-(23,111),4,BF
260 LINE(24,42)-(75,111),4,BF
270 LINE(64,64)-(103,111),7,BF
280 LINE(104,88)-(159,111),5,BF
290 LINE(168,80)-(239,111),13,BF
300 LINE(175,104)-(231,111),1,B
400                    'strasse zeichnen
410 LINE(0,112)-(255,191),14,BF
420 LINE(0,121)-(255,122),15,BF
430 LINE(0,176)-(255,179),15,BF
440 LINE(0,137)-(255,139),15,BF
900 GOTO 900           'schleife, damit bild am schirm bleibt
```

Lassen Sie dieses Modul mit dem Befehl RUN <RETURN> laufen. Haben Sie das Programm nicht sehr sorgfältig abgetippt, findet der MSX-ASIC-Interpreter unter Umständen Syntax-Fehler. In Anhang B finden Sie die MSX-BASIC-Fehlercodes und -Meldungen, die Ihnen bei der Fehlersuche behilflich sein können. Korrigieren Sie die Fehler und versuchen Sie es noch einmal. Ist alles in Ordnung, zeichnet das Modul eine Straßenszene, wie Sie sie in Bild 8.1.1 sehen.

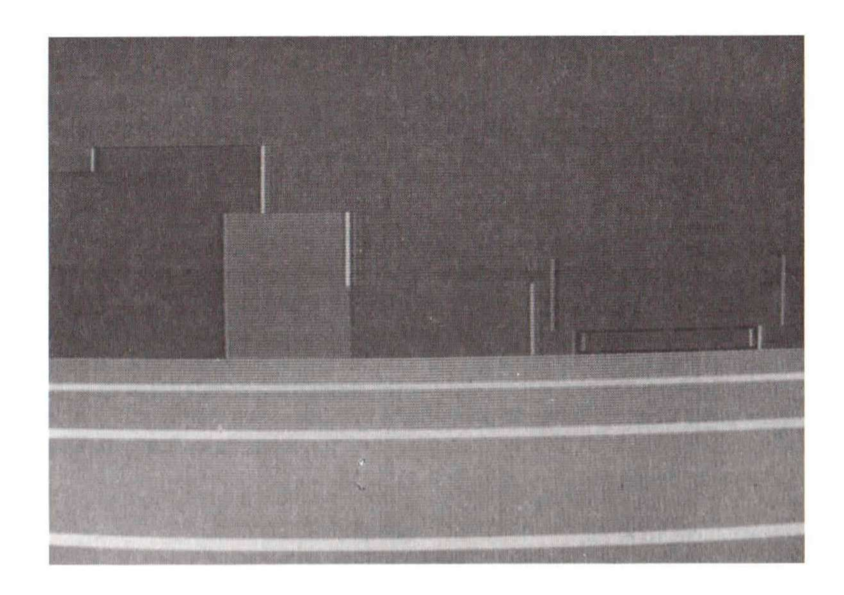


Bild 8.1.1 SZENE1

Drücken Sie <CTRL>+<STOP>, um die Programmausführung zu unterbrechen. Wie in Zeile 100 angegeben, ändert sich die Hintergrundfarbe in Rot. Wollen Sie sie Blau haben, müssen Sie folgenden Befehl eingeben:

```
COLOR 15,4 <RETURN>
```

Drücken Sie <SHIFT>+<F1>, erhalten Sie das gleiche Ergebnis. Beachten Sie, daß die Zeile:

```
900 GOTO 900
```

eine Dauerschleife ist, die das Programm immer weiter laufen läßt. Sie sorgt dafür, daß das Bild am Schirm erhalten bleibt. Wird diese Zeile vergessen, kehrt MSX-BASIC in den Befehlsmodus zurück und löscht das Bild.

Nennen wir dieses Modul "SZENE1" und legen es mit SAVE im ASCII-Format für spätere Anwendungen ab. Verwenden Sie eine neue Kassette. Spulen Sie sie zurück und setzen Sie den Recorder-Zähler auf 0. Stellen Sie den Recorder auf Aufnahme und geben Sie den Befehl:

```
SAVE "CAS:SZENE1" <RETURN>
```

ein. Können Sie sich erinnern, daß Sie diesen Befehl in Kapitel 6 gelernt haben?

Der Vollständigkeit halber wollen wir die relative Form von LINE auch erklären. STEP (X-Offset, Y-Offset) geht vom letzten Punkt aus. In einer LINE-Anweisung bezieht sich das Argument auf die erste Koordinate, wenn die relative Form verwendet wird. Ein Beispiel:

```
10 PSET(10,10)
20 LINE STEP(20,20)-STEP(30,30)
```

In diesem Fall wird ein Punkt bei den Koordinaten (10,10) gesetzt. Der STEP-Offset (20,20), der sich auf den letzten Punkt von 10,10 bezieht, bestimmt, daß der Befehl LINE bei (30,30) beginnt. Der LINE-Befehl veranlaßt, daß eine Linie von (30,30) nach (60,60) gezogen wird.

Läßt man den Farbcode weg:

```
LINE(0,0)-(10,10),,B
```

wird die Vordergrundfarbe genommen. Wird im LINE-Befehl eine Koordinate verwendet, die außerhalb des Bereichs liegt, wird sie auf den größt- beziehungsweise kleinstmöglichen Wert gesetzt.

Negative Werte werden zu Null.
X-Werte größer als 256 werden 256
Y-Werte größer als 192 werden 192

8.2 Die Anweisungen CIRCLE und PAINT

Die CIRCLE-Anweisung zeichnet Kreise oder Ellipsen entsprechend der angegebenen Mittelpunkts- und Radiuswerte. Das Format lautet:

Format 1:

```
CIRCLE (X,Y),<Radius>[, <Farbe>][, <Start>][, <Ende>][, <Aspekt>]
```

Format 2:

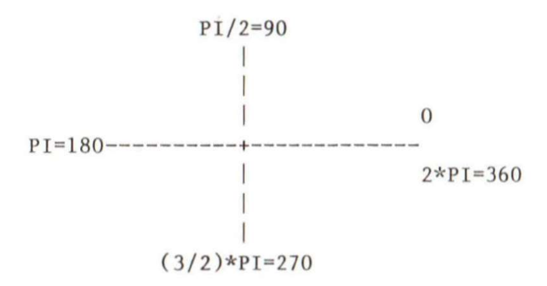
```
CIRCLE STEP(X,Y),<Radius>[,<Farbe>][,<Start>][,<Ende>][,<Aspekt>]
```

Das erste Argument (X,Y) oder STEP(X,Y) gibt den Mittelpunkt des Kreises an. Im ersten Format werden absolute Koordinaten angegeben. Das zweite bezieht sich auf den letzten Referenzpunkt.

‹Radius› ist die Entfernung vom Mittelpunkt bis zum Rand des Kreises.

‹Farbe› ist ein optionales Argument, das die Farbe des Kreises bestimmt. Wird sie nicht angegeben, wird die Vordergrundfarbe eingesetzt.

Die ‹Start›- und ‹Ende›-Argumente geben Winkel zwischen 0 und 2*PI (ungefähr 6,28) in Radiant an, die bestimmen, wo die Zeichnung beginnt und wo sie endet. Das folgende Bild zeigt diese Zuweisung:



Sind der Start- oder Endpunkt negativ, wird der Kreis mit dem Mittelpunkt durch eine Linie verbunden. Die Winkel selbst werden als positiv angenommen. Ein Kuchendiagramm wird gezeichnet.

‹Aspekt› ist die Beziehung des waagrechten Maßstabs zum senkrechten Maßstab. Wird der Wert nicht mit angegeben, ist er eins. Andere Werte ergeben vertikal oder horizontal liegende Elipsen.

Bevor wir den CIRCLE-Befehl einsetzen, lernen wir etwas über die PAINT-Anweisung.

Die PAINT-Anweisung füllt jede grafische Figur mit der Farbe aus, die Sie angeben. Begonnen wird dabei von den angegebenen Koordinaten, bis eine angegebene Farbgrenze erreicht wird. Das Format lautet:

```
PAINT (X-Start, Y-Start)[, <Füllfarbe>][, <Randfarbe>]
```

Die Koordinate, an der der Zeichenvorgang beginnen soll, kann absolut oder relativ zum letzten Referenzpunkt angegeben werden, darf aber nicht außerhalb des Bildbereichs liegen. Damit der Befehl ausgeführt wird, dürfen die Koordinaten von PAINT nicht auf der Begrenzungslinie der auszufüllenden Figur liegen.

PAINT verwendet die Vordergrundfarbe, wenn die ‹Füllfarbe› weggelassen wird. Als ‹Randfarbe› wird die ‹Füllfarbe› gesetzt, wenn es nicht anders angegeben ist. Im Grafik-Modus mit der hohen Auflösung (SCREEN 2) kann keine ‹Randfarbe› gesetzt werden. Das ist deshalb so, weil im SCREEN-2-Modus die ‹Füllfarbe› als ‹Randfarbe› betrachtet wird.

Jetzt wollen wir unsere Straßenszene um einen Baum ergänzen und dafür CIRCLE und PAINT verwenden. Löschen Sie den Speicher des Computers. Erinnern Sie sich noch an den NEW-Befehl? Tippen Sie das Programm-Modul ab. Haben Sie die Programmbeispielkassette, laden Sie "SZNE2M" von Seite A:

```
100 COLOR 15,9,1
110 SCREEN 2,3
510                                'baum zeichnen
520 CIRCLE(28,28),22,2,,,2         'umrisse der blaetter
530 PAINT(28,80),2                 'blaetter zeichnen
540 LINE(24,100)-(31,115),6,BF     'baumstamm zeichnen
900 GOTO 900
```

Lassen Sie dieses Modul laufen und beseitigen Sie eventuelle Fehler. Sie erhalten einen gezeichneten Baum, wie Sie ihn in Bild 8.2.1 sehen.

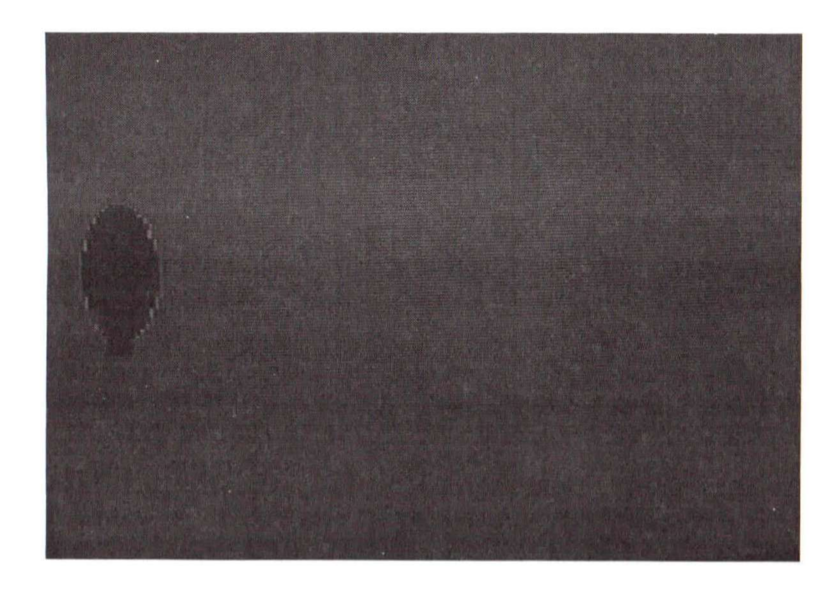


Bild 8.2.1 SZNE2M

Wieder müssen Sie <CTRL>+<STOP> drücken, um die Ausführung anzuhalten. Um die Straßenszene einzubinden, hängen wir mit MERGE "SZENE1" das gespeicherte Programm von der Kassette an das Modul:

1. Spulen Sie die Kassette bis zum Anfang von "SZENE1" zurück.

2. Setzen Sie den Kassetten-Recorder in den Wiedergabemodus.

3. Geben Sie folgenden Befehl ein:

```
MERGE "CAS:SZENE1" <RETURN>
```

4. Wenn die Ok-Bereitschaftsanzeige erscheint, geben Sie RUN <RETURN> ein. Die Straße, die Gebäude und der Baum erscheinen am Bildschirm. Gefällt es Ihnen? (Bild 8.2.2)

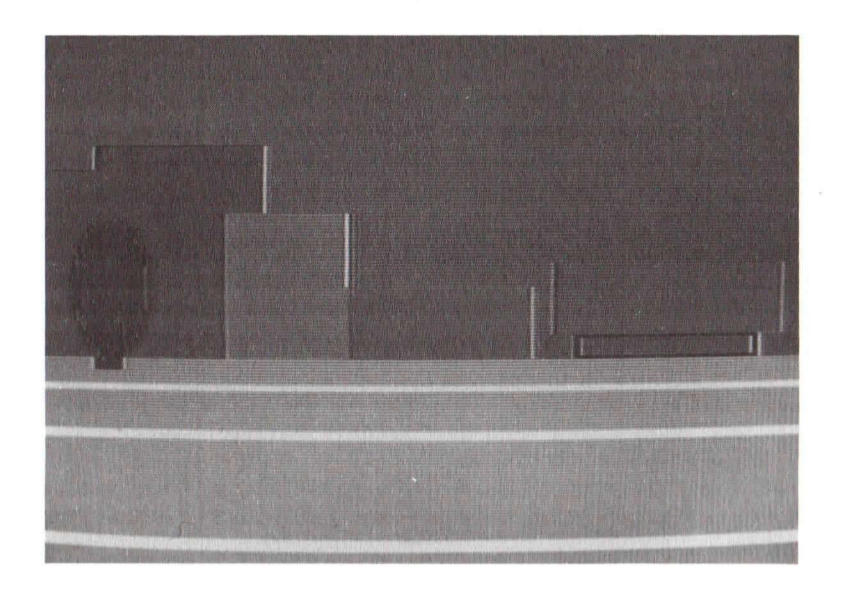


Bild 8.2.2 SZENE2

Speichern Sie dieses zusammengehängte Modul als "SZENE2" im ASCII-Format auf Ihrer Kassette ab.

8.3 Die DRAW-Anweisung

Die DRAW-Anweisung verbindet Grafikanweisungen mit leicht zu handhabenden Unterkommandos, die man Grafik-Makro-Sprache nennt (GML = Graphic macro language).

Ein GML-Befehl besteht aus einem einfachen Zeichen innerhalb einer Zeichenkette. Während der Ausführung überprüft MSX-BASIC die Werte der Zeichenkette und interpretiert die einzelnen Buchstaben als Befehle der Zeichenkette. Das Format von DRAW lautet:

```
DRAW <"Zeichenkettenausdruck">
```

Der Zeichenkettenausdruck kann folgendermaßen aussehen:

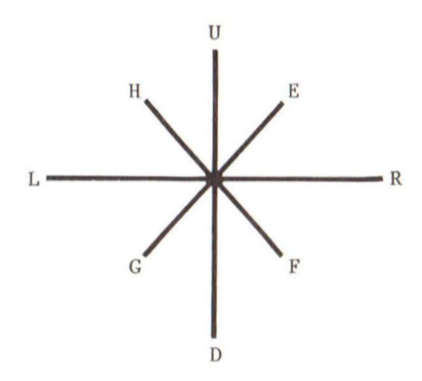
```
10 SCREEN 2
20 PSET(100,100)
30 A$="D2R2U2L2"      'GML-Zeichenkette definieren
40 DRAW A$            'Die Anweisung zeichnen
50 GOTO 50
```

Auf diese Weise können Sie die Sequenz an einer anderen Stelle in Ihrem Programm verwenden, ohne den gesamten DRAW-Befehl noch einmal eingeben zu müssen. DRAW beginnt am letzten Referenzpunkt. Wird er mit CIRCLE eingesetzt, beginnt er im Mittelpunkt des Kreises.

8.3.1 GML-Unterbefehle

Jeder der folgenden GML-Unterbefehle beginnt vom letzten Referenzpunkt an, der mit einem anderen Befehl wie LINE, PSET oder einer anderen BASIC-Grafikanweisungen angegeben wurde.

U[<n>]	n*(Teilungsfaktor) Punkte nach oben
D[<n>]	nach unten
L[<n>]	nach links
R[<n>]	nach rechts
E[<n>]	diagonal nach oben und rechts
F[<n>]	diagonal nach unten und rechts
G[<n>]	diagonal nach unten und links
H[<n>]	diagonal nach oben und links



‹n› mit dem Skalierungsfaktor multipliziert ergibt die Anzahl der bewegten Punkte (siehe Abschnitt 8.3.5). Wird ‹n› nicht angegeben, werden die Punkte nur um eine Pixelgröße bewegt. Hierzu ein Beispiel:

```
10 SCREEN 2
20 PSET(100,100)
30 A$="U10"
40 DRAW A$
50 GOTO 50
```

Das Programm zeichnet eine senkrechte Linie 10 Punkte nach oben.

8.3.2 Absolutes und relatives Zeichnen

DRAW kann mit dem GML-Befehl in absoluter und relativer Form verwendet werden:

```
M(X,Y)
```

Steht ein Plus- oder ein Minuszeichen vor X, werden X und Y zu der laufenden Grafikposition dazuaddiert und mit der momentanen Position durch eine Linie verbunden. Sind Plus- oder Minuszeichen nicht vorhanden, wird eine Linie von der momentanen Position nach (X,Y) gezogen. Ein Beispiel:

```
10 CLS                    'Bildschirm löschen
20 SCREEN 2               'hochauflösender Grafikmodus
30 LINE(100,0)-(100,50)   'zieht eine senkrechte Linie
40 A$="M25,25 U10"        'GML-Befehlszeichenkette
50 DRAW A$
60 GOTO 60
```

Dieses Beispiel zieht Linien vom Punkt (100,0) nach (100,50), von (100,50) nach (25,25) und von (25,25) nach (25,15). Alle Adressen sind absolut angegeben.

Wird Zeile 40 in

```
40 A$="M+25,25 U10"
```

geändert, wird die zweite Linie von (100,50) nach (125,75) gezogen und der letzte Punkt liegt dann bei (125,65).

Wird Zeile 40 in:

```
40 A$="M-25,25 U10"
```

geändert, wird die zweite Linie von (100,50) nach (75,75) gezogen. Wo liegt dann der letzte Punkt? (Er ist bei (75,65))

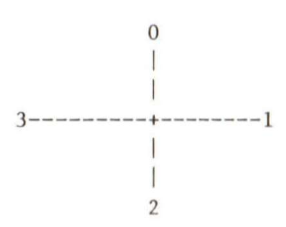
Die Plus- und Minuszeichen zeigen die relativen Startpunkte an.

8.3.3 Winkel

Der GML-Befehl:

```
A[ ‹n› ]
```

setzt Winkel (engl. Angle). ‹n› liegt im Bereich von 0 bis 3, wobei 0 = 0 Grad, 1 = 90 Grad, 2 = 180 Grad und 3 = 270 Grad entspricht. Die Richtung entspricht dem Uhrzeigersinn, was das folgende Bild zeigt:



Dies wird mit einem Beispiel verdeutlicht:

```
10 CLS
20 SCREEN 2
30 LINE(100,0)-(100,50)
40 LINE(0,50)-(100,50)
50 A$="U10 R30 D10 L30"
60 DRAW A$
70 GOTO 70
```

Dieses Programm zeichnet eine X,Y-Achse und ein Rechteck rechts davon. Fügen wir den Winkelbefehl in Zeile 50 hinzu:

```
50 A$="A0 U10 R10 D10 L30"
```

wird das Rechteck unverändert gezeichnet, weil Null der Ausgangslage entspricht. Würde jedoch

```
50 A$="A1 U10 R30 D10 L30"
```

eingegeben, würde das Rechteck im Uhrzeigersinn um 90 Grad gedreht. Spielen Sie ein bißchen mit A2 und A3 in Zeile 50, damit Sie diesen Befehl kennenlernen. Denken Sie daran, den Winkel wieder auf den Standardwert (0) zurückzusetzen, wenn Sie weiter programmieren wollen.

MSX-BASIC hält den letzten eingegebenen Winkelwert fest und benutzt ihn für alle weiteren Angaben.

8.3.4 Präfix-Befehle

Mit dem Präfix B können Sie an verschiedene Positionen des Bildschirms gelangen, ohne Punkte zu zeichnen. Ein Beispiel:

```
10 CLS
20 SCREEN 2
30 LINE(100,0)-(100,50)
40 A$="B M75,25 U10"
50 DRAW A$
60 GOTO 60
```

Dieses Programm zieht eine Linie von (100,0) nach (100,50) und eine Linie von (75,25) nach (75,15). Die Linie, die mit M75,25 erzeugt würde, wird nicht gezeichnet.

Der Präfix N bewegt den Cursor auf die Originalposition. Lautet Zeile 40:

```
40 A$="N M75,25 U10 R20"
```

wird M75,25 gezeichnet und der Cursor kehrt nach (100,50) zurück. Dann bewegt er sich 10 Punkte nach (100,40) und zeichnet dort eine Linie nach (120,40).

Mit dem C‹n›-Präfix wird die Farbe gesetzt. ‹n› muß im Bereich von 0 bis 15 liegen. Wird:

```
40 A$="N M75,25 U10 C6 R20"
```

eingegeben, wird die letzte Zeile in dunkelroter Farbe gezeichnet.

8.3.5 Der Skalierbefehl

Der Skalierungsbefehl:

```
S ‹n›
```

vergrößert oder verkleinert eine Figur. Der Skalierungsfaktor wird mit den Entfernungen, die durch die Befehle U, D, L, R oder M gegeben sind, multipliziert und man erhält die tatsächliche Entfernung. ‹n› liegt im Bereich zwischen 0 und 255. Der Standardwert ist 0. Beide Ausdrücke S0 und S4 entsprechen der Originalgröße. Wollen Sie die Figur verkleinern, wählen Sie S1 bis S3. Größere Werte geben (n/4)-mal die Originalgröße.

Bevor die Programmierung weiter fortgesetzt wird, muß der Skalierungsbefehl S0 übergeben werden. MSX-BASIC merkt sich den zuletzt eingegebenen Skalierungsfaktor und arbeitet damit bei allen nachfolgenden Entwürfen.

8.3.6 Die Ausführung von Teilzeichenketten

Der Befehl:

```
X ‹Zeichenketten-Variable›
```

erlaubt es, den zweiten Teilstring einer Zeichenkette auszuführen. Ein Beispiel:

```
10 CLS
20 SCREEN 2
30 PSET(50,50),8
40 A$="R10 D10 L10 U10"
50 DRAW "S1XA$;S10XA$;S20XA$;S50XA$;"
60 GOTO 60
```

Dieses Programm führt die Teilzeichenkette A$ in vier Größen aus. Beachten Sie, daß das Semikolon (;) im X-Befehl notwendig ist.

Ein Hinweis:
In allen GML-Befehlen können die Argumente ‹n›, X oder Y Konstanten oder numerische Variable sein. Wird eine Variable verwendet, muß ein Istgleichzeichen (=) davorstehen und ein Semikolon (;) folgen. Leerräume werden in GML-Befehlszeichenketten ignoriert.

Jetzt wollen wir in unsere Straßenszene Berge zeichnen. Löschen Sie den Speicher des Computers und geben Sie folgendes Programm-Modul ein:

```
100 COLOR 15,9,1
110 SCREEN 2,3
200                              'Berge zeichnen
205 A$="A0BM255,95C3L104U32E20R3E10R5F10"
210 B$="R1F2R2F3R5E2R3E5R1E20R1E2R2E1R5F1R1"
215 DRAW"XA$;XB$;"
220 PAINT(250,94),3
225 C$="BM255,111C2L168U40E25U1E3R8E2R3F1R1F5D1F5R2"
230 D$="R1F10R1F11R3F8R19E16U1E2R1E2R5F1R5F3R5E9R3E3R5"
235 DRAW"XC$;XD$;"
240 PAINT(250,110),2
245 GOTO 245
```

Mit diesem Programm-Modul werden zwei Berge gezeichnet, wie Sie es in Bild 8.3.6.1 sehen. Sie finden dieses Programm unter dem Namen "SZNE3M" auf der beim Verlag erhältlichen Programmbeispielkassette.

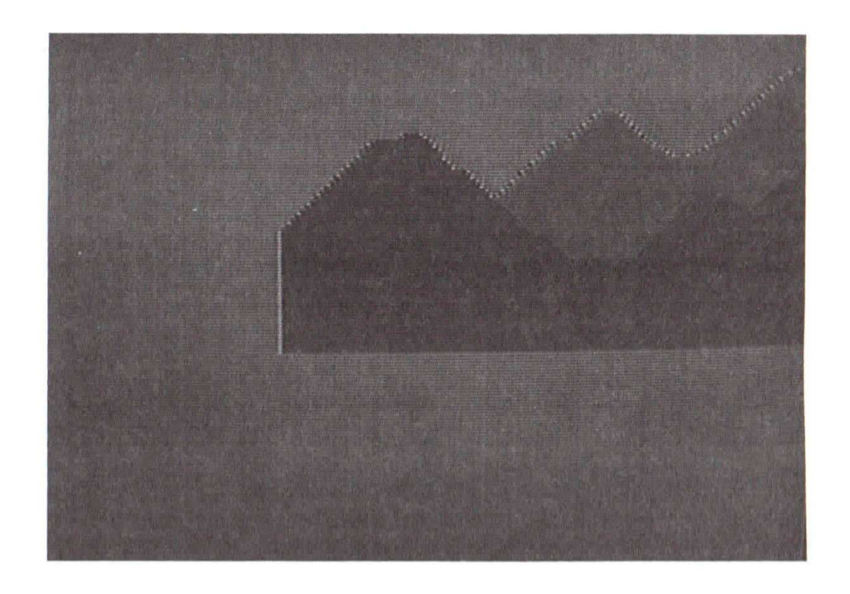


Bild 8.3.6.1 SZNE3M

Läuft das Programm fehlerfrei, hängen Sie mit MERGE das Programm "SZENE2" von der Kassette dazu. Das Programm zeichnet Gebäude, Bäume und Berge (Bild 8.3.6.2). Speichern Sie dieses Programm als "SZENE3" auf der Kassette, damit Sie es später noch einsetzen können.

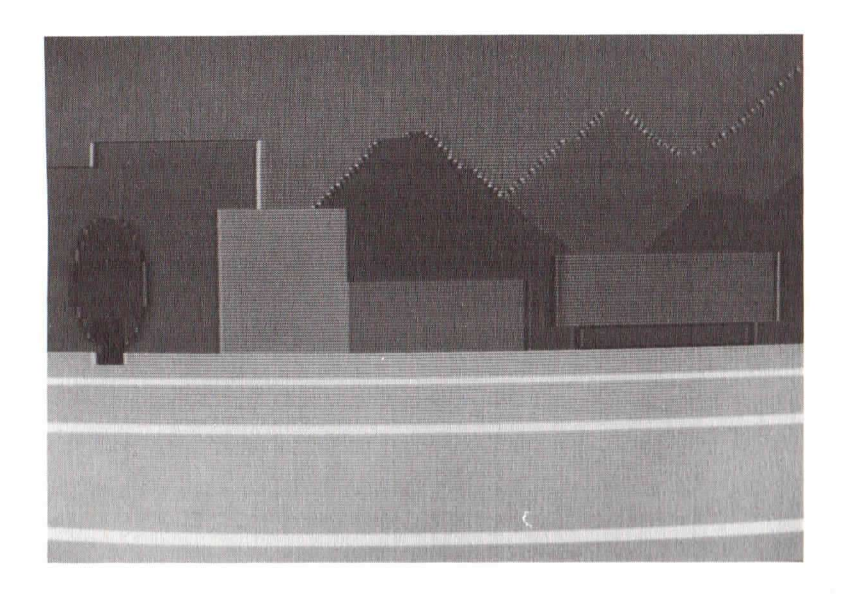


Bild 8.3.6.2 SZENE3

Im nächsten Kapitel erklären wir, wie man bewegliche Objekte in die Straßenszene einbringen kann.

8.4 BASIC-Wortschatz

Hier finden Sie eine Liste der MSX-BASIC-Wörter, die in diesem Kapitel erklärt worden sind:

LINE	8.1
GOTO	8.1
CIRCLE	8.2
PAINT	8.2
DRAW	8.3

9

Bewegliche Bilder

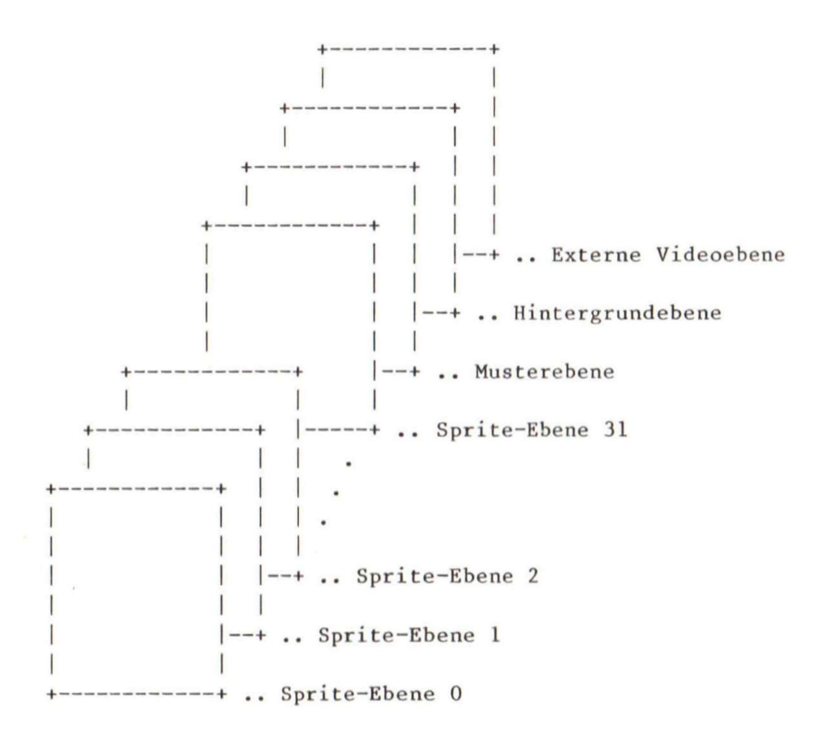


9 Bewegliche Bilder

Nachdem Sie gelernt haben, wie man nette Bilder auf dem Bildschirm zeichnet, wollen wir nun neue Befehle, mit denen man bewegte Bilder erzeugen kann, kennenlernen. Das MSX-BASIC hält dafür entsprechende Anweisungen bereit. Zuerst müssen wir jedoch das Konzept der Sprites und die Möglichkeiten des Video-Display- Prozessors (VDP) kennenlernen.

Sie wissen schon, daß der VDP vier Bildschirm-Modi (SCREEN 0-3) besitzt.

Der VDP gibt ein Bild auf den Schirm aus, das sich aus 35 verschiedenen Ebenen zusammensetzt:



Die Objekte auf den Ebenen, die dem Betrachter am nächsten sind, haben die höchste Priorität. Befinden sich also zwei Figuren auf zwei verschiedenen Ebenen an der gleichen Stelle am Bildschirm, wird die Figur mit der höchsten Priorität gezeigt. Soll eine

Figur an einer besonderen Ebene dominant sein, müssen alle Ebenen davor transparent, das heißt, nicht beschrieben sein.

Sprites bestehen aus beweglichen Mustern, deren Position am Bildschirm durch die waagrechten und senkrechten Koordinaten, die im Videospeicher (VRAM) abgelegt sind, definiert ist. Die ersten 32 Ebenen (Spriteebene 0-31) können jeweils einen Sprite enthalten. Aus diesem Grund können 32 Sprites zur gleichen Zeit am Bildschirm erscheinen. Die Gebiete in den Spriteebenen, die nicht aus den Sprites selbst bestehen, sind transparent. Die Koordinaten der Sprites bestehen aus Pixelangaben. Deshalb können Sprites sehr genau positioniert und bewegt werden. Sprites gibt es in drei verschiedenen Größen: 8 x 8 Pixels, 16 x 16 Pixels und 32 x 32 Pixels. In einer waagrechten Linie können nur 4 Sprites aktiv sein. Befinden sich auf einer Linie weitere Sprites, werden sie automatisch in den Transparent-Modus gesetzt. Im Textmodus (SCREEN 0) stehen keine Sprites zur Verfügung.

Hinter den Sprite-Ebenen befindet sich die Musterebene. Die Musterebene wird für Text- und Grafikbilder verwendet. Die Befehle LINE, CIRCLE, PAINT und DRAW finden hier Anwendung.

Hinter der Musterebene liegt die Hintergrundebene. Sie ist größer als die anderen Ebenen, weshalb man einen Rahmen um die anderen Ebenen herum erstellen kann. Mit der COLOR-Anweisung kann man die Rahmenfarbe auswählen.

Die letzte Ebene mit der niedrigsten Priorität ist die externe Videoebene. Bilder für diese Ebene kommen aus einem externen Videosignal. Die externe Video-Schnittstelle erlaubt das Mischen einer externen Videoquelle mit dem VDP-Signal, so daß ein Bild unter Software-Kontrolle erstellt werden kann. Auf diese Weise können Bilder aus den Muster- und Spriteebenen mit einem hereinkommenden Signal gemischt werden. Diese Schnittstelle ist nicht bei allen MSX-Geräten vorhanden.

Um ein Sprite im MSX-BASIC zu erstellen, müssen Sie:

1. ein Sprite auswählen, indem Sie SPRITE$ verwenden. Zur gleichen Zeit können Sie am Bildschirm maximal 32 Sprites darstellen.

2. den Sprite-Umriß definieren. Dafür müssen Sie die BASIC-Anweisungen READ, DATA, CHR$ und VAL kennen.

3. die Spritefarbe auswählen.

4. das Sprite auf dem Bildschirm mit PUT SPRITE ausgeben.

In den folgenden Abschnitten werden wir diese Anweisungen genauer kennenlernen.

9.1 Die FOR...NEXT-Anweisung

Schleifen werden gewöhnlich dann eingesetzt, wenn man eine Reihe von Anweisungen mehrere Male wiederholen muß. Die FOR...NEXT-Anweisungen durchlaufen mehrere Male eine Programmschleife und verlassen diese Schleife, wenn die angegebene Anzahl der Schleifendurchläufe erreicht ist. Das allgemeine Format lautet:

```
FOR ‹Variable› = X TO Y [STEP Z]
.
.
.
NEXT [‹Variable›][, ‹Variable›...]
```

‹Variable› ist der Schleifenzähler. Der erste Zahlenausdruck (X) ist der Anfangswert des Zählers. Der 2. Zahlenausdruck (Y) ist der Endwert. Der 3. Zahlenausdruck (Z) ist die Schritt-Größe (STEP = Schritt), mit der gezählt wird. Die Programmzeilen, die nach FOR stehen, werden ausgeführt, bis NEXT erreicht wird. Dann wird der Zähler um den in Z stehenden Wert erhöht. Ist kein Wert angegeben, wird als Standardwert +1 dazuaddiert. Der laufende Zählerstand wird überprüft. Ist er niedriger als der Endwert, wird der Prozeß fortgesetzt. Ist er größer, fährt das Programm mit der nächsten Anweisung, die nach NEXT steht, fort. Ein Beispiel:

```
10 FOR I = 1 TO 10
20 PRINT I
30 NEXT I
```

Das Programm gibt die Zahlen von 1 bis 10 auf den Bildschirm aus. In diesem Fall ist STEP nicht angegeben. Ist STEP negativ, muß der Endwert kleiner sein als der Anfangswert, weil dann der Zähler nach jedem Schleifendurchlauf kleiner wird.

FOR...NEXT-Schleifen können verschachtelt werden. Eine FOR...NEXT-Schleife kann also in eine andere FOR...NEXT-Schleife eingebunden werden. Sind sie verschachtelt, muß jede FOR...NEXT- Schleife einen einheitlichen Variablen-Namen als Zähler besitzen. Die NEXT-Anweisung der inneren Schleife muß vor der NEXT-Anweisung der äußeren Schleife erscheinen. Haben verschachtelte Schleifen den gleichen Endpunkt, reicht ein einziges NEXT für beide Schleifen aus.

Wird die Variable nach NEXT nicht angegeben, wird die NEXT-Anweisung der zuletzt angegebenen FOR-Anweisung zugeordnet.

Wird NEXT durchlaufen, bevor es über den Befehl FOR kommt, wird eine "NEXT without FOR"-Fehlermeldung ausgegeben. Erscheint FOR ohne NEXT, meldet sich der

Computer mit "FOR without NEXT". In beiden Fällen unterbricht MSX-BASIC die Ausführung des Programms und kehrt in den Befehlsmodus zurück.

9.2 Die Anweisungen READ und DATA

Eine READ-Anweisung muß immer in Verbindung mit einer DATA-Anweisung stehen. Die Werte in einer DATA-Anweisung werden den Variablen, die in den READ-Anweisungen stehen, direkt zugeordnet. Die Werte in den DATA-Anweisungen müssen mit dem Variablen-Typ in den READ-Anweisungen übereinstimmen, sonst wird ein Syntax-Fehler gemeldet.

Mit der DEF-Typzuweisung werden Variablen, die mit speziellen Buchstaben beginnen, einem bestimmten Variablen-Typ zugeordnet. Wird keine Zuweisung vorgenommen, setzt MSX-BASIC alle Variablen als Variablen mit doppelter Genauigkeit (Abschnitt 4.3.) ein. Ein Beispiel:

DEFDBL A-D	Alle Variablen, die mit dem Buchstaben A, B, C und D beginnen, sind Variablen mit doppelter Genauigkeit.
DEFINT E-F	Alle Variablen, die mit dem Buchstaben E und F beginnen, sind Ganzzahl-Variablen.
DEFSNG G-K	Alle Variablen, die mit dem Buchstaben G, H, I, J und K beginnen, sind Variablen mit einfacher Genauigkeit.
DEFSTR X-Z	Alle Variablen, die mit dem Buchstaben X, Y und Z beginnen, sind Zeichenketten-Variablen.

Sie erinnern sich sicher, daß Zeichenketten-Variablen einfache Variablen darstellen, die aus einer Reihe von Zeichen bestehen. Zusätzlich kennt MSX-BASIC mehrere Funktionen, mit denen Variablen-Typen manipuliert werden können. Einige davon werden wir für die Sprite-Definition einsetzen:

FUNKTION	AKTION
ASC(X$)	Gibt einen numerischen Wert zurück, der dem ASCII-Code des ersten Zeichens der Zeichenkette X$ entspricht.
CHR$(X)	Gibt ein Zeichen zurück, das dem ASCII-Code für den Wert X entspricht.
STR$(X)	Gibt eine Zeichenkette als Repräsentation des Wertes von X zurück.
VAL(X$)	Gibt den numerischen Wert der Zeichenkette X$ zurück.

Um numerische und Zeichenketten-Konstanten für den fortlaufenden Zugriff der READ-Anweisung abzuspeichern, müssen wir den nicht ausführbaren DATA-Befehl anwenden:

```
DATA <Konstantenliste>
```

DATA-Anweisungen können überall im Programm placiert werden. Sie können soviele Konstanten beinhalten, wie in einer Zeile Platz finden. Die Konstanten müssen mit Kommas getrennt werden. Im Programm können beliebige DATA-Anweisungen eingesetzt werden.

Die <Konstantenliste> kann numerische Konstanten in jedem beliebigen Format enthalten. Zeichenketten-Konstanten müssen nur dann durch Anführungszeichen abgetrennt werden, wenn sie Kommas oder Doppelpunkte enthalten.

Die READ-Anweisung, die die Werte aus der DATA-Anweisung liest und den Variablen zuordnet, muß folgendes Format haben:

```
READ <Variablenliste>
```

Eine einzige READ-Anweisung kann auf eine oder mehrere DATA-Anweisungen zugreifen. Es ist auch erlaubt, daß mehrere READ-Anweisungen die gleiche DATA-Anweisung benutzen. Übersteigt die Anzahl der Variablen in der <Variablenliste> die Anzahl der Elemente in den DATA-Anweisungen, erfolgt eine "Out of DATA"-Fehlermeldung. Ist die Anzahl der Variablen in der <Variablenliste> kleiner als die Anzahl der DATA-Elemente, lesen nachfolgende READ-Anweisungen das erste noch nicht gelesene DATA-Element. Gibt es kein nachfolgendes READ, werden die Daten ignoriert.

Mit der RESTORE-Anweisung können die DATA-Anweisungen neu initialisiert werden, so daß sie wieder von Anfang an gelesen werden können.

Im nächsten Abschnitt wollen wir mit den neuen Befehlen einfache Sprites erstellen.

9.3 Die Anweisungen SPRITE$ und PUT SPRITE

Nun wollen wir ein Sprite auf den Bildschirm ausgeben. Tippen Sie das nächste Programm ein und lassen Sie es laufen:

```
10 CLS                    'Bildschirm loeschen
20 SCREEN 2,0             'Grafikmodus und 8x8 Sprite waehlen
30 FOR I=1 TO 8           'FOR...NEXT-Schleife starten
40 READ A$                'DATA-lesen
50 S$=S$+CHR$(VAL("&B"+A$))
60 NEXT I                 'Ende der FOR...NEXT-Schleife
70 SPRITE$(1)=S$          'Muster Nr. 1 zuordnen
80 PUT SPRITE 0,(100,100),15,1     'auf Ebene 0 Sprite Nr.1
90                                 'bei (100,100) ausgeben
100 GOTO 100
110 DATA 00011000                  'Sprite-Daten
120 DATA 00111100                  '0 ergeben transparente
130 DATA 01111110                  'Punkte, 1 sind sicht-
140 DATA 11111111                  'bare Punkte
150 DATA 00011000
160 DATA 00011000
170 DATA 01100110
180 DATA 01100110
```

Dieses Programm zeigt einen weißen Pfeil (100,100), wie Sie ihn in Bild 9.3.1 sehen. In diesem Fall haben wir binäre Zahlen verwendet, um den Sprite-Umriß festzulegen. (Das binäre Zahlensystem finden Sie in Anhang C). Experimentieren Sie mit verschiedenen Sprite-Umrissen und geben Sie sie an den unterschiedlichsten Positionen auf den Bildschirm aus. Es wird Ihnen Spaß machen.

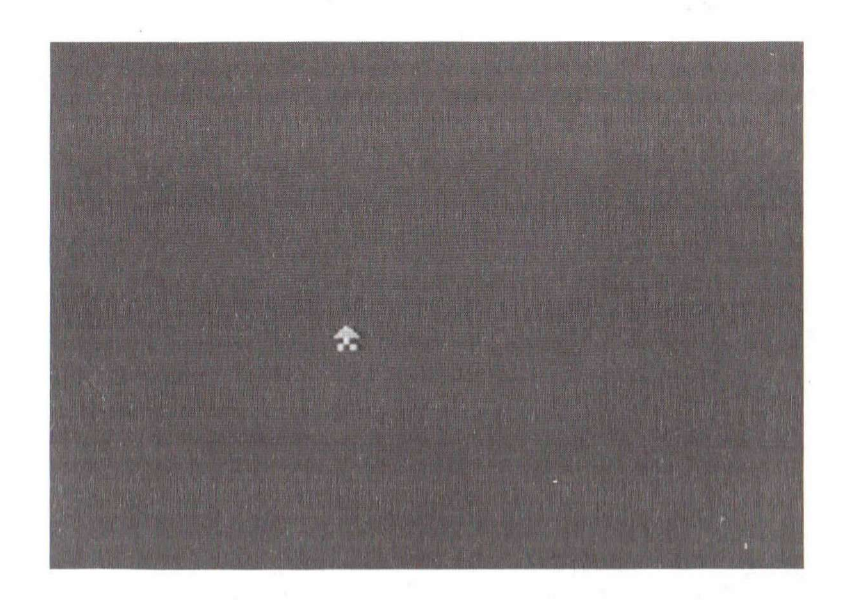


Bild 9.3.1 Ein Sprite

Das allgemeine Format für die Definition der Muster oder Umrisse eines Sprites lautet:

```
SPRITE$ <Sprite-Musternummer>
```

Die <Sprite-Musternummer> muß kleiner sein als 256, wenn die Spritegröße, die mit der SCREEN-Anweisung ausgewählt wurde, 0 oder 1 ist. Ist die Spritegröße 2 oder 3, muß sie sogar kleiner sein als 64.

Die beiden Formate, mit denen man Sprites auf den Bildschirm ausgibt, sehen so aus:

Format 1:

```
PUT SPRITE <Ebenen-Nummer>[, (X,Y)][,<Farbe>][,<Musternummer>]
```

Format 2:

```
PUT SPRITE <Ebenen-Nummer>[, STEP (X-Offset, Y-Offset)]
        [, <Farbe>][, <Musternummer>]
```

‹Ebenen-Nummer› muß im Bereich zwischen 0 und 31 liegen.

Die beiden Formate unterscheiden sich nur in der Koordinaten-Spezifikation. Format 1 verwendet absolute Koordinaten, während Format 2 Offset-Werte vom letzten Referenzpunkt ab, der durch einen MSX-BASIC-Grafikbefehl gegeben werden kann, einsetzt. Es muß darauf hingewiesen werden, daß im Vergleich zur Pixelgrafik für die Sprite-Koordinaten ein vertikaler Offset von -1 zu berücksichtigen ist. Deshalb müssen die Koordinaten (0,-1) verwendet werden, wenn ein Sprite an die linke obere Ecke des Bildschirms mit den Pixel-Koordinaten (0,0) gesetzt werden soll.

Die ‹Musternummer› spezifiziert die Muster der Sprites. Betrachten Sie auch den SPRITE$-Befehl. Wird diese Nummer nicht mit angegeben, wird die ‹Musternummer› der ‹Ebenen-Nummer› zugeordnet.

Um zu testen, ob Sie alles verstanden haben, wollen wir ein größeres Sprite auf dem Bildschirm ausgeben:

```
10 CLS
20 SCREEN 1,3                               '16x16 Spritegroesse waehlen
30 FOR I=1 TO 16                            'Beginn FOR...NEXT-Schleife
40 READ D$                                  'DATA fuer Sprites
50 A$=A$+CHR$(VAL("&B"+LEFT$(D$,8)))        'linke Seite lesen
60 B$=B$+CHR$(VAL("&B"+RIGHT$(D$,8)))       'rechte Seite lesen
70 NEXT I                                   'Ende der FOR...NEXT-Schleife
80 SPRITE$(1)=A$+B$                         'Muster Nr.1 waehlen
90 PUT SPRITE 1, (130,110),3,1
100 PUT SPRITE 2, (120,100),2,1
110 PUT SPRITE 3, (100,125),10,1
120 GOTO 120
130 DATA 0000000010000000         'Sprite-Daten
140 DATA 0000000010000000         'Nullen sind transparent
150 DATA 0000000111000000         'Einsen sind sichtbar
160 DATA 0000001111100000         'zuerst muessen die 8 linken
170 DATA 0000011111100000         'Spalten gelesen werden,
180 DATA 0000010111110000         'dann die 8 rechten Spalten
190 DATA 0000111111110000
200 DATA 0000111111111000
210 DATA 0001111111111000
220 DATA 0001111111111000
230 DATA 0011011111111100
240 DATA 0010111111110100
250 DATA 0011011111101100
260 DATA 0001111110111000
270 DATA 0000111111110000
280 DATA 0000000000000000
```

Dieses Programm zeichnet drei Birnen mit unterschiedlichen Farben (Bild 9.3.2). Beachten Sie, daß der Computer das 16x16-Sprite so aufbaut, daß er zuerst die linke Seite mit 8 x 16 und dann die rechte Seite mit 8 x 16 auffüllt. Wir benutzen die Funktion LEFT$(D$,8) in Zeile 50, um die 8 linken Zeichen von D$ in A$ zu lesen. Entsprechend gibt RIGHT$(D$,I) die rechten I-Zeichen von D$ zurück.

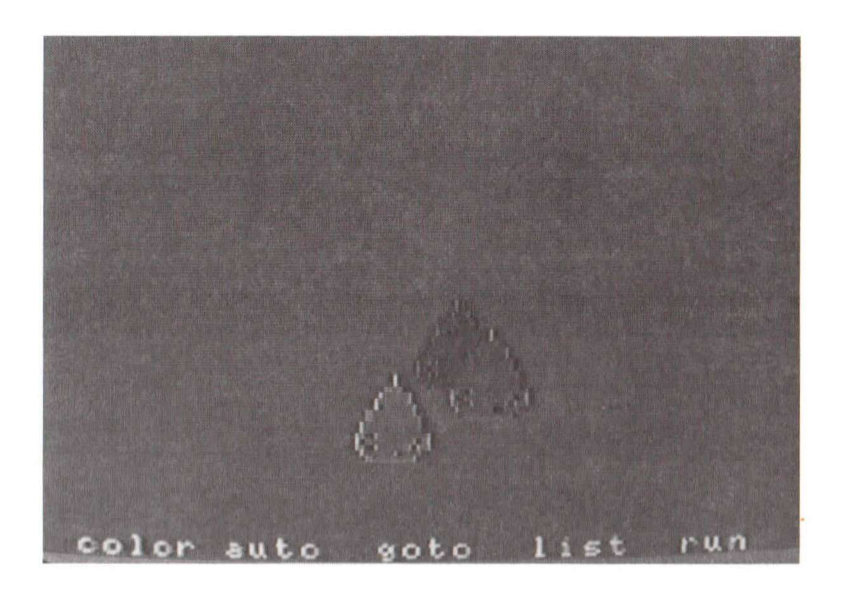


Bild 9.3.2 Drei Birnen

Während die binäre Darstellung ein Abbild des Sprites darstellt, wird es schwierig, wenn wir die Daten für mehrere Sprites benötigen. Im hexadezimalen System (Anhang C) stellen wir vier binäre Stellen mit einer hexadezimalen Stelle dar. Für das vorangegangene Beispiel würde dies so aussehen:

```
40 READ D$
50 A$=A$+CHR$(VAL("&H"+D$))
55 READ D$
60 B$=B$+CHR$(VAL("&H"+D$))
130 DATA 00,80,00,80,01,C0,03,E0
140 DATA 07,E0,05,F0,0F,F0,0F,F8
150 DATA 1F,F8,1F,F8,37,FC,2F,F4
160 DATA 37,EC,1F,B8,0F,F0,00,00
```

Die Zeilen 170 - 280 brauchen wir nicht mehr. Wahrscheinlich ist Ihnen der kompaktere Code lieber. Im nächsten Beispiel benutzen wir das hexadezimale Zahlensystem. Wenn Sie damit noch nicht zurechtkommen, sehen Sie bitte im Anhang C nach.

9.4 Wie man eine Figur bewegt

Nun wissen Sie, wie man Sprites erstellt. Versuchen wir, sie zu bewegen und lassen wir sie über den Bildschirm laufen.

Am einfachsten gestaltet man ein Programm, wenn man zuerst einen Entwurf anfertigt, in dem festgelegt wird, was das Programm tun soll. Stellen Sie sich vor, ein Auto soll am unteren Bildschirmrand entlang fahren. Der Entwurf dazu könnte so aussehen:

1. Wir definieren den Umriß des Autos.

2. Wir definieren die Räder.

3. Wir geben das Bild auf den Bildschirm aus.

4. Wir lassen das Bild über den Bildschirm laufen.

Nun übersetzen wir den Entwurf in eine Bildsprache, die im wesentlichen aus Schlagwörtern besteht und schon BASIC-Wörter enthält. Diese Sprache nennen wir Pseudo-Code. Der Pseudo-Code für unseren Entwurf:

1. COLOR Vordergrund, Hintergrund und Rand

2. Auswahl des Grafik-SCREEN-Modus

3. Mit READ Daten für den Autorumpf lesen.

4. Mit READ Daten für die Räder lesen.

5. Das SPRITE auf den Bildschirm ausgeben.

6. Das SPRITE über den Bildschirm bewegen.

Der Pseudo-Code zeigt uns, daß wir für das Lesen der Daten des Autos und der Räder das gleiche Unterprogramm benutzen können. Er zeigt auch, daß wir die BASIC-Anweisungen, die wir für dieses Programm benötigen, im letzten Abschnitt gelernt haben. Jetzt ist es ganz einfach, den Pseudo-Code in ein BASIC-Programm

umzusetzen. Geben Sie das Programm-Modul ein. Haben Sie die zum Buch erhältliche Programmbeispielkassette, können Sie das Modul "SZNE4M" von Seite A laden:

```
100 COLOR 15,9,1                 'Farben auswaehlen
110 SCREEN 2,3                   'Schirmmodus ,SPRITE-Groesse
125 S=1:GOSUB 1000               'Daten fuer Autorumpf
130 S=2:GOSUB 1000               'Daten fuer Raeder
570 PUT SPRITE 3,(148,107),8,1     'geparktes Auto
580 PUT SPRITE 4,(148,107),1,2
590 FOR X=0 TO 255                 'FOR...NEXT-Schleife
610 PUT SPRITE 1,(255-X,130),2,1   'Auto ueberquert
620 PUT SPRITE 2,(255-X,130),1,2   'Bildschirm
630 NEXT X                         'Ende FOR...NEXT-Schleife
650 GOTO 590
1000                          'Unterprogramm fuer Spritedef.
1010 S$=""                    'S$ string auf Null setzen
1020 FORJ=1 TO 32             'Beginn FOR...NEXT-Schleife zum Dateneinlesen
1030 READ D$
1040 S$=S$+CHR$(VAL("&H"+D$))
1050 NEXT J
1060 SPRITE$(S)=S$
1070 RETURN
3000                             'Daten des Autos
3010 DATA OF,OF,OF,09,3E,22,E2,C3  'links oben
3020 DATA FF,FF,E7,43,43,00,00,00  'links unten
3030 DATA FF,FF,FF,FF,FF,FF,81,FF  'rechts oben
3040 DATA FF,FF,F3,E1,E1,00,00,00  'rechts unten
4000                             'Daten der Autoreifen
4010 DATA 00,00,00,00,00,00,00,00  'links oben
4020 DATA 00,00,1B,3C,3C,18,00,00  'links unten
4030 DATA 00,00,00,00,00,00,00,00  'rechts oben
4040 DATA 00,00,0C,1E,1E,0C,00,00  'rechts unten
```

Dieses Programm zeichnet ein geparktes Auto. Dann fährt das Auto am unteren Rand über den Bildschirm. Beachten Sie, daß das Unterprogramm in den Zeilen 1000 bis 1070 verwendet wird, um sowohl das Auto, als auch die Räder des Autos festzulegen. Damit sparen wir Platz. Beachten Sie auch, daß die Daten in hexadezimaler Form eingegeben werden. Warum? Erinnern wir uns, daß zwei hexadezimale Stellen 8 Binärstellen repräsentieren. Wenn wir hexadezimale Zahlen verwenden, ersparen Sie sich eine Menge Tipparbeit. Am besten entwerfen Sie Ihre Spritemuster zuerst auf einem 8 x 8 karierten Papier.

In den Zeilen 590 bis 630 wird das Auto über den Bildschirm bewegt, weil die linken oberen Koordinaten der Sprites geändert werden. Wie kann man die Geschwindigkeit des Autos steigern? Sie können die Schrittweite (STEP) in der FOR...NEXT-Schleife ändern. Versuchen Sie es!

Hängen sie nun "SZENE3" von der Kassette an dieses Programm-Modul an. Jetzt sehen Sie ein Auto über Ihre Straßenszene rollen (Bild 9.4.1). Wenn Sie schon Erfahrung mit anderen Computern haben, werden Sie die einfache Handhabung für das Erstellen bewegter Bilder in MSX-BASIC zu schätzen wissen. Speichern Sie das Programm als "SZENE4" auf Ihrer Kassette ab. "SZENE4" befindet sich auch auf der Programmbeispielkassette.

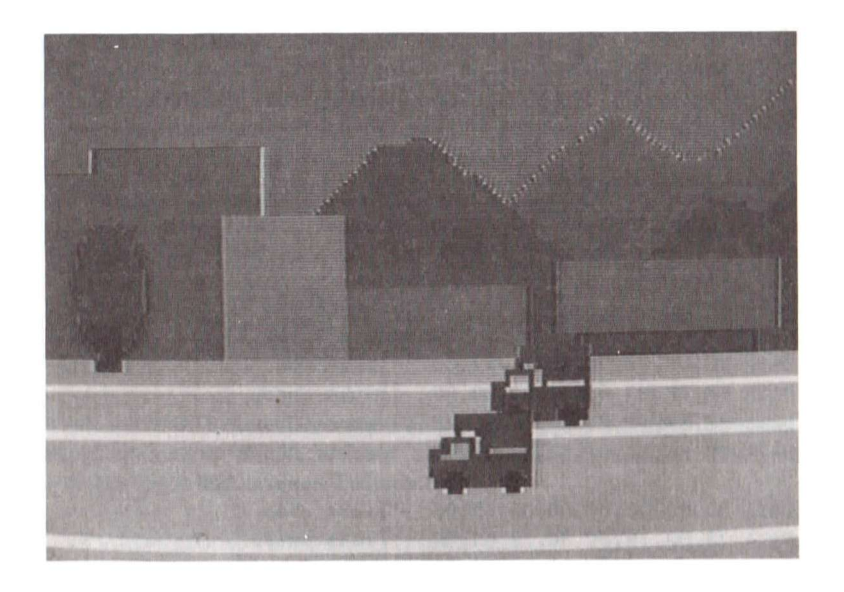


Bild 9.4.1 SZENE4

9.5 Der Abschluß

Nehmen wir an, Sie wollen etwas in Ihre Straßenszene hineinschreiben. Das geht mit dem PRINT#-Befehl. Er lautet in der allgemeinen Form:

```
PRINT #<Dateinummer>,<Ausdruck>
```

Bevor irgendeine Ein- oder Ausgabe mit PRINT # gemacht werden kann, muß ein Kanal mit der OPEN-Anweisung geöffnet werden. Damit wird "GRP:" als Bildschirmausgabe definiert. OPEN hat das allgemeine Format:

```
OPEN "<Gerätezuweisung>[<Dateinummer>]"[FOR<Modus>] AS
     [#] <Dateinummer>
```

Die OPEN-Anweisung bereitet einen Puffer für die Ein/Ausgabe vor und setzt den Modus, in dem er verwendet wird.

Die Gerätezuweisungen, die für den weiteren Ablauf geöffnet werden müssen, heißen:

CAS: Kassetten-Recorder
CRT: Bildschirm
GRP: Grafik-Bildschirm
LPT: Drucker

Um also Zeichen auf den grafischen Bildschirm unserer Straßenszene zu zeichnen, müssen wir folgende BASIC-Anweisungen verwenden:

```
115 OPEN "GRP:" AS #1     'Grafikschirm + Puffer Nr.1 oeffnen
560 PSET(32,48),4         'Cursor setzen, Farbe waehlen
562 PRINT#1,"M T L"       '"M T L" auf Gebaeude schreiben
563 PSET(185,93),13       'Cursor erneut setzen
565 PRINT#1,"M S X"       '"M S X" auf Reklametafel schreiben
```

Fügen Sie diese Zeilen zu "SZENE4" und freuen Sie sich über Ihre neue Schöpfung (Bild 9.5.1). Im nächsten Kapitel reichern wir unsere Straßenszene mit Musik an. Speichern Sie Ihr Programm nun als "SZENE5" ab.

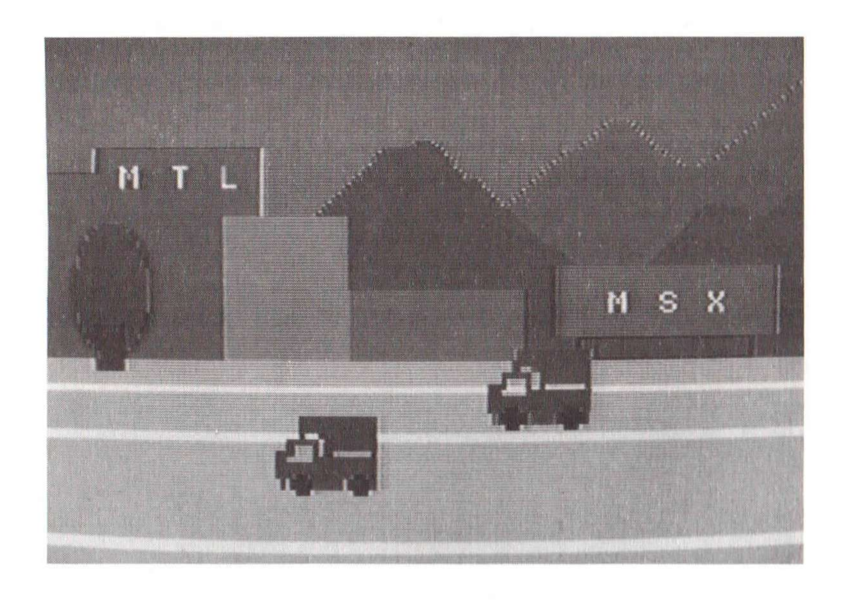


Bild 9.5.1 SZENE5

9.6 BASIC-Wortschatz

Anschließend finden Sie eine Liste der in diesem Kapitel verwendeten BASIC-Befehle:

FOR...NEXT	9.1
DEFDBL	9.2
DEFINT	9.2
DEFSNG	9.2
DEFSTR	9.2
ASC	9.2
CHR$	9.2
STR$	9.2
VAL	9.2
READ	9.2
DATA	9.2
RESTORE	9.2
CLS	9.3
SPRITE$	9.3
PUT SPRITE	9.3
LEFT$	9.3
RIGHT$	9.3

10

Musik



10 Musik

Erst mit Musik und anderen Geräuschen machen Video-Spiele so richtig Spaß. Der einfachste Befehl, mit dem man Töne erzeugt lautet:

```
BEEP
```

Ein Piepston dieser Art kann bei den meisten Computern durch die Ausgabe CHR$(7) erzeugt werden.

MSX-BASIC kann noch viel mehr! Es erlaubt, Musik zu komponieren. Sie können die Tonhöhe, die Länge der Noten und andere Tonvariationen frei programmieren.

Ihr MSX-Computer enthält einen Synthesizer, den man als programmierbaren Tongenerator (engl. Programmable Sound Generator = PSG) bezeichnet. Er kann seine Arbeit unabhängig vom Hauptprozessor erledigen. Mit MSX-BASIC können Sie Töne oder Musik erzeugen, ohne den Tongenerator laufend zu bedienen. Sind die Initialisierungswerte einmal gegeben, kann der PSG selbständig weiter Töne produzieren.

Benutzt man immer nur eine Note mit einmal, kann man sehr leicht passable Töne erklingen lassen. Die Erzeugung von Akkorden steigert die Tonqualität weiter. Der PSG hat drei unabhängig voneinander steuerbare Kanäle, mit denen man richtige Musikstücke programmieren kann.

Um die Qualität der Darbietung noch zu steigern, kann MSX-BASIC volle acht Oktaven erzeugen. Die unabhängige, programmierbare Amplitudensteuerung für jeden Kanal erlaubt einen Tonbereich von 16 Ebenen, vorausgesetzt, Sie verwenden den prozessorgesteuerten Amplitudenmodus.

Die Steuerbefehle werden dem PSG übergeben, indem man in die 16 Register schreibt. Ein Register spricht den PSG an, während zwei andere Ein-/Ausgabeports sind. Damit bleiben 13 Register für die Ton-Steuerung übrig.

Auf den ersten Blick mag das oben Gesagte etwas verwirren. Aber keine Angst, es gibt zwei einfache Befehle, nämlich PLAY und SOUND, mit denen man den Tongenerator bedienen kann.

10.1 Die PLAY-Anweisung

Der Befehl PLAY arbeitet vom Konzept her genauso wie DRAW. Er verwendet eine Musik-Makro-Sprache (engl. Music Macro Language = MML), die in einer Befehlszeichenkette abgelegt wird.

Ein MML-Befehl ist ein einzelnes Zeichen innerhalb einer Zeichenkette. Während der Ausführung interpretiert MSX-BASIC das einzelne Zeichen als Befehl. Wird eine Nullzeichenkette eingeben, bleibt der Stimmkanal ausgeschaltet. Das Format von PLAY lautet:

```
PLAY <Zeichenkettenausdruck für Stimme A>
     [, <Zeichenkettenausdruck für Stimme B>]
     [, <Zeichenkettenausdruck für Stimme C>]
```

Der Zeichenkettenausdruck kann folgendermaßen aussehen:

```
10 A$="CDEFGAB"
20 PLAY A$
30 GOTO 20
```

Lassen Sie dieses Programm laufen und erzeugen Sie mit Ihrem MSX-Computer Töne. Wir werden verschiedene MML-Befehle in den nächsten Abschnitten beschreiben.

10.1.1 MML-Befehle

Die einzelnen Zeichenbefehle für den Zeichenkettenausdruck in PLAY sind die Buchstaben, die die Musiknoten von A-G darstellen. PLAY spielt einfach die angezeigte Note in der entsprechenden Oktave. Die optionalen Zeichen "#" oder "+" erhöhen den Ton um eine halbe Note (A+=Ais), während das Zeichen "-" einen um eine halbe Note tieferen Ton erzeugt (A-=As, was den schwarzen Tasten einer Klaviertastatur entspricht).

```
+----------------------------------------------
|   |   | |   |   |   |   | |   | |   |   |   |
|   |C+ | |D+ |   |   |F+ | |G+ | |A+ |   |   |
|   |D- | |E- |   |   |G- | |A- | |B- |   |   |
|   |   | |   |   |   |   | |   | |   |   |   |
|   +---+ +---+   |   +---+ +---+ +---+   |   +--
|     |     |     |     |     |     |     |     |
|  C  |  D  |  E  |  F  |  G  |  A  |  B  |  C  |
|     |     |     |     |     |     |     |     |
+------------------------------------------------
```

E#, F-, B+ und C- sind nicht erlaubt. Um alle drei PSG-Kanäle für die Erzeugung von Akkorden zu aktivieren, muß der Zeichenkettenausdruck mit Kommas getrennt werden. Ein Beispiel:

```
10 A$ = "CEG"        'erste Zeichenkette definieren
20 B$ = "EAB"        'zweite Zeichenkette definieren
30 C$ = "GCD"        'dritte Zeichenkette definieren
40 PLAY A$           'ersten String spielen
50 PLAY B$           'zweiten String spielen
60 PLAY C$           'dritten String spielen
70 PLAY A$,E$,C$     'alle drei gleichzeitig spielen
80 GOTO 40
```

Zuletzt hören Sie, wie drei Noten zusammen gespielt werden. Sie können jeden Kanal so programmieren, daß jeder etwas anderes spielt und damit Melodien oder Harmonien erzeugen.

10.1.2 Präfix-Befehle

Alle MML-Präfix-Befehle bleiben so lange aktiv, bis sie durch einen anderen Präfix geändert werden. Damit Sie die Wirkung einiger Präfix-Befehle kennenlernen, lassen wir ein einfaches Beispielprogramm laufen:

```
10 A$="CDE"
20 PLAY A$
30 GOTO 20
```

O‹n›

Der Präfix "O‹n›" (Oktaven) setzt die Oktave für die nachfolgenden Noten. Es gibt 8 Oktaven, die von 1 (niedrigste) bis 8 (höchste) numeriert sind. Jede Oktave geht von "C" bis "B". Oktave O4 ist die Standardoktave. Geben Sie die nächste Zeile ein und lassen Sie das obige Programm nocheinmal laufen:

```
10 A$="O1CDEO4CDEO8CDE" 'von der niedrigsten zur höchsten
```

Sie hören, wie die Noten in drei verschiedenen Oktaven gespielt werden. MSX-BASIC merkt sich die zuletzt eingegebene Oktave. Solange sie nicht geändert wird, werden alle Noten gespielt.

L‹n›

Der Präfix "L‹n›" (Länge) bestimmt die Länge der Noten. Die zu n gehörenden Notenlängen sind in der folgenden Tabelle zusammengefaßt:

LÄNGE	ENTSPRICHT
L1	eine ganze Note (am längsten)
L2	eine halbe Note
L3	ein Drittel einer Note
L4	eine Viertelnote
L5	ein Fünftel einer Note
.	
.	
L64	ein 64stel einer Note (am kürzesten)

Die aktuelle Länge ist 1/n eines Viertakt-Schlages. ‹n› liegt im Bereich von 1 bis 64. Der Standardwert ist 4. Testen Sie Ihr Programm mit folgender Zeile:

```
10 A$="L1CDEL4CDEL64CDE"
```

Die Länge einer Einzelnote kann auch als nachfolgende Zahl stehen. So ist A16 das gleiche wie L16A.

Steht ein Punkt (.) nach einer Note, muß die Länge mit 3/2 multipliziert werden (die Länge der Note verlängert sich um das 1/2fache). Stehen mehrere Punkte hinter einer Note, wird die Länge jeweils pro Punkt um die Hälfte des vorher addierten Werts verlängert. Ein Beispiel soll das verdeutlichen:

```
A$="A...."      'Die Länge der Note: 1+1/2+1/4+1/8+1/16 = 31/16
```

R<n>

Der Präfix "R<n>" (Rest = Pause) setzt die Längen der Pausen zwischen zwei Noten. Er wird ähnlich behandelt wie "L<n>". <n> liegt im Bereich von 1 bis 64. Der Standardwert ist 4. Ändern Sie Zeile 10 in:

```
10 A$="L4CDER1CDER4CDER64CDE"
```

Beobachten Sie, wie sich die Pausenlängen verändern.

T<n>

Der Präfix "T<n>" (Tempo) setzt die Anzahl der Viertelnoten pro Minute. Ändern Sie Zeile 10 in:

```
10 A$="L4T32CDET120CDET255CDE"
```

Sie hören, wie die Noten in drei verschiedenen Stufen gespielt werden. <n> liegt im Bereich von 32 - 255. Der Standardwert ist 120.

V<n>

Der Präfix "V<n>" steuert die Lautstärke. <n> liegt im Bereich von 0 bis 15. Der Standardwert ist 8. Die Zahl 15 bedeutet größte Lautstärke. Betrachten Sie die nächste Zeile:

```
10 A$="V0CDEV8CDEV15CDE"
```

Sie hören, wie die nachfolgenden Noten immer lauter werden.

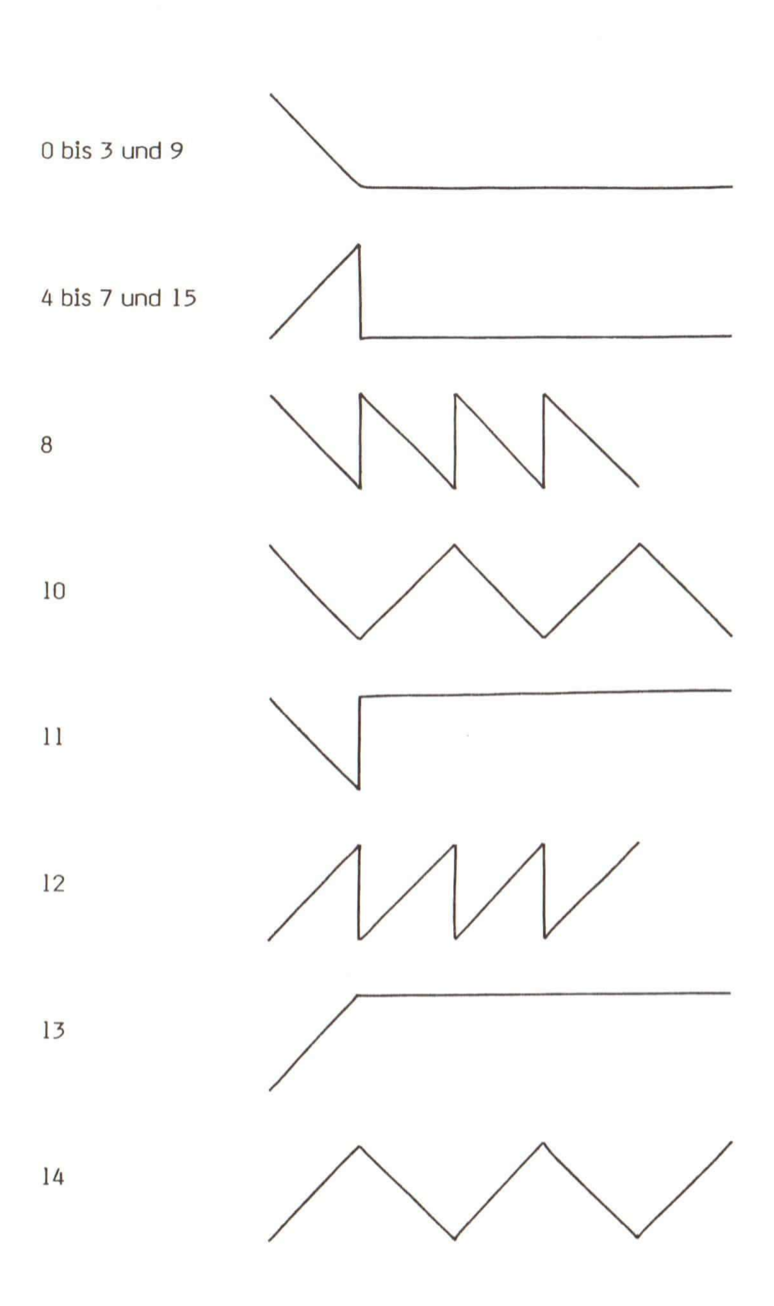
M<n>

Der Präfix "M<n>" setzt die Modulation (betrachten Sie auch Präfix "S<n>"). <n> liegt im Bereich von 1 bis 65535. Der Standardwert ist 255.

S<n>

Der Präfix "S<n>" setzt die Hüllkurve (Klangform des Tons). <n> liegt im Bereich von 0 bis 15. Folgende Hüllkurvenformen gibt es:

<n>	Hüllkurvenform



Diese Kurven könnte man als die Stimmen des PSG ansehen. Versuchen Sie folgende Zeile:

```
10 A$="S1CDES4CDES8CDES10CDES11CDES12CDES13CDE"
```

Hören Sie sich die unterschiedlichen Stimmen der unterschiedlichen Hüllkurven an.

Ist die Hüllkurve einmal gesetzt, kann man mit dem Präfix M‹n› die Stimmen verändern, indem man die Dauer ändert:

```
10 A$="S0M500CDEM5000CDE"
```

Hören Sie sich die unterschiedlichen Stimmen an, die bei der gleichen Hüllkurve mit unterschiedlicher Dauer erklingen.

10.1.3 So kann man Noten und Oktaven auch anwählen

MSX-BASIC stellt neben der Oktaven- (O‹n›) plus Namensanwahl (A bis G) noch einen anderen Weg zur Verfügung, mit dem man die Noten anwählen kann. Der Befehl lautet:

N‹n›

‹n› liegt im Bereich von 0 bis 96 und gibt direkt den Notenwert an. ‹n›=0 bedeutet Ruhepause. O4C entspricht dem Wert ‹n›=36. Wir verwenden diesen Befehl im Beispiel des nächsten Kapitels.

10.1.4 Die Ausführung eines Unterkommandos

Mit dem Befehl:

X ‹Zeichenkettenvariable›

kann ein Unterkommando, das einer Zeichenkettenvariablen zugewiesen wurde, ausgeführt werden. Die Ähnlichkeit mit den Unterkommandos in der DRAW-Anweisung ist nicht zu übersehen. Ein Beispiel:

```
10 A$="CDE"
20 PLAY "O3XA$;O5XA$;O7XA$;"
30 END
```

Dieses Beispiel spielt A$ dreimal in drei Oktaven. Der Strichpunkt (;) ist für den X-Befehl notwendig.

In allen MML-Befehlen kann das Argument <n> eine Konstante oder eine numerische Variable sein. Wird eine <Variable> verwendet, muß ihr ein Istgleichzeichen (=) voranstehen und ein Semikolon (;) folgen. Ein Beispiel:

```
10 FOR I=0 TO 96
20 PLAY "N=I;"
30 NEXT I
40 GOTO 10
```

Beachten Sie, daß alle Werte von <n>, die in MML-Befehlen spezifiziert wurden, auf den Standardwert zurückgesetzt werden, wenn der Befehl BEEP verwendet wird.

10.2 Die Straßenszene mit Musik

Nun wollen wir mit der PLAY-Anweisung Musik in unsere Straßenszene "SZENE5" bringen. Löschen Sie den Speicher des Computers und geben Sie die nächsten Zeilen ein. (Unter dem Namen "SZNE6M" finden Sie das Programm auch auf der Programmbeispielkassette.)

```
581                            'Musik strings fuer PLAY definieren
582 M$="V15T180M4000S0L805C04G05CEEG06C05G06EDC"  'string 1
583 N$="05GL2GL806FFDFEEDCD05BGB06L4CL8EG07L2C"   'string 2
584 O$=M$ + N$                 'string fuer Kanal A zusammensetzen
585 P$="V6T180M4000S0L804GCEGCEG05C04GBECCGEG"
586 Q$="04G05D04B05DCG04B05G04G05D04B05D04G05GEL4G"
587 R$=P$ + Q$                 'string fuer Kanal B zusammensetzen
590 X=1                        'diese Zeile wird ersetzt
600 IF X=1 THEN PLAY O$,R$     'mit PLAY zwei Kanaele bedienen
```

Läßt man das Programmsegment alleine laufen, spielt es eine kleine Melodie und zeigt nichts am Bildschirm. Wird "SZENE5" mit MERGE von der Kassette dazugeladen, beginnt die Musik zu spielen, wenn das Auto an der rechten Seite des Bildschirms erscheint. Musik und Aktion sind damit synchronisiert.

Speichern Sie dieses Programm als "SZENE6" ab, damit Sie es Ihren Freunden zeigen können.

Die Möglichkeiten von PLAY sind nahezu unbegrenzt. Es gibt viele unterschiedliche Arten, Musik zu erzeugen. Deshalb ist es recht schwierig allgemeingültige Aussagen dazu zu machen. In diesem Bereich können Sie Ihrer Phantasie freien Lauf lassen.

10.3 Die SOUND-Anweisung

Anders als bei der PLAY-Anweisung, mit der Sie Musik erzeugen können, kontrollieren Sie mit der SOUND-Anweisung den PSG selbst. Mit SOUND schreiben Sie Werte direkt in die Register des PSG. Überspringen Sie diesen Abschnitt, wenn Sie glauben, daß er für Sie zu kompliziert ist.

Die allgemeine Form von SOUND lautet:

```
SOUND ‹PSG-Register›, ‹gewünschter Wert›
```

‹PSG-Register› ist eines der verfügbaren PSG-Register und ‹gewünschter Wert› kann eine Zahl von 1 bis 255 sein. Das folgende PSG-Blockdiagramm beschreibt die Funktionen der verschiedenen Register:

Register	Bit	B7	B6	B5	B4	B3	B2	B1	B0
R0	Kanal A Tonperiode	untere 8 Bit (Ton A)							
R1						obere 4 Bit (A)			
R2	Kanal B Tonperiode	untere 8 Bit (Ton B)							
R3						obere 4 Bit (B)			
R4	Kanal C Tonperiode	untere 8 Bit (Ton C)							
R5						obere 4 Bit (C)			
R6	Geräusch	5 Bit Geräuschperiode							

Register	Bit	B7	B6	B5	B4	B3	B2	B1	B0
R7	Zuordnung	ein/aus		Geräusch			Ton		
		10B	10A	C	B	A	C	B	A
R8	Amplitude A				M	L3	L2	L1	L0
R9	Amplitude B				M	L3	L2	L1	L0
R10	Amplitude C				M	L3	L2	L1	L0
R11	Länge der Hüllkurve	untere 8 Bit (Hüllkurvenperiode)							
R12		obere 8 Bit (Hüllkurvenperiode)							
R13	Hüllkurve					E3	E2	E1	E0
R14	E/A-Port A	8 Bit Parallelport A							
R15	E/A-Port B	8 Bit Parallelport B							

Hierzu eine einfache Erklärung:

Register	Funktionen und Operationen
R0-R5	Tongenerator-Steuerung
R6	Geräuschgenerator-Steuerung
R7	Wählt die einzelnen Kanäle aus
R8-R10	Amplitudensteuerung
R11-R13	Hüllgenerator-Steuerung

Der PSG hat drei Tonkanäle: A, B und C. Die Registerpaare steuern die Frequenz eines jeden Kanals, man nennt Sie HIGH- und LOW-Register. Sie sind nachfolgend aufgeführt:

KANAL	HIGH-Register	LOW-Register
A	1	0
B	3	2
C	5	4

Den ‹gewünschten Wert› erhält man, wenn man den Eingangstakt durch die 16fache gewünschte Frequenz dividiert. Ein Beispiel:

‹gewünschter Wert› = 1789773/(16 x gewünschte Frequenz)

LOW-Register = ‹gewünschter Wert› AND 255

HIGH-Register = ‹gewünschter Wert›/256

Die gewünschte Frequenz wird in Zyklen pro Sekunde oder Hertz angegeben. Hier ist ein Beispiel für die Verwendung des Kanals A:

```
10 INPUT "GEBEN SIE DIE FREQUENZ EIN:";Z
20 D=1789773/(16*Z)       'gewuenschten Wert berechnen
30 L=D AND 255            'Wert fuer LOW-Register
40 H=D/256                'Wert fuer HIGH-Register
50 SOUND 0,L              'Kanal A LOW-Register
60 SOUND 1,H              'Kanal A HIGH-Register
70 SOUND 7,&B11111110     'Ton fuer Kanal A einschalten
80 I=15
90 SOUND 8,I              'Kanal A Lautstärke
100 GOTO 10
```

Dieses Beispielprogramm erzeugt die Frequenz, die Sie über die Tastatur eingeben. Die Lautstärke wird in Zeile 90 eingestellt. Folgende Register steuern die die Lautstärke der Kanäle:

Kanal	Register
A	8
B	9
C	10

Der ‹gewünschte Wert› für diese Register liegt im Bereich von 0 bis 15. Fügen Sie die nächsten Zeilen zum obigen Programm hinzu:

```
85 FOR  I=15 TO 0 STEP-1        'Beginn FOR...NEXT-Schleife
95 FOR  J=1 TO 100: NEXT J      'Zeitschleife
96 NEXT I                       'Ende FOR...NEXT-Schleife
```

Sie hören einen langsam abklingenden Ton, wenn die Lautstärke des Kanals A von 15 nach 0 heruntergezählt wird.

Zusätzlich kann das Lautstärken-Steuerregister 8 die Amplituden- Kontrolle an die Hüllkurvenlängen- und Hüllkurven-Steuerregister 11, 12 und 13 übergeben. Damit das geschieht, muß der Wert des Registers auf 16 gesetzt sein.

Das Register 7 spricht die drei Geräusch- und Tonkanäle an. Damit Geräusch- und Tonfrequenzen gemischt werden, müssen die Bits 0 bis 5 des Registers 7 folgendermaßen gesetzt sein:

B7	B6	B5	B4	B3	B2	B1	B0
ohne Bedeutg		Geräuschkanal			Tonkanal		
		C	B	A	C	B	A

Die Bits B7 und B6 werden für Ein- und Ausgabeports verwendet und haben für MSX-BASIC keine Bedeutung. Die Bits B3 und B0 steuern Kanal A usw. Der logische Wert 0 gibt den Kanal frei, während 1 den Kanal rücksetzt:

```
SOUND 7,&B11111110 'spricht Tonkanal A an
SOUND 7,&B11110110 'spricht Ton- und Geräuschkanal A an
```

Wie in der PLAY-Anweisung können mit dem SOUND-Befehl komplizierte Hüllkurvenmuster erzeugt werden. Die Register 11 und 12 steuern die Hüllkurvenfrequenz, während das Register 13 die Kurvenform bestimmt. Die Register 11 und 12 müssen als 16 Bit-Register betrachtet werden. Das Setzen der Hüllkurvenfrequenz entspricht der des Frequenzsetzens für jeden Kanal. Ein Beispiel:

‹gewünschter Hüllkurvenwert› = 1789773/(256*Frequenz)

LOW-Register 11 = ‹gewünschter Hüllkurvenwert› AND 255

HIGH-Register 12 = ‹gewünschter Hüllkurvenwert›/256

Die ‹Frequenz› muß in Hertz angegeben werden. Mit Register 13 können Sie 8 Hüllkurvenmuster ausgeben. Das nächste Beispiel zeigt diese Befehle:

```
10 SOUND 0,100              'Kanal A LOW-Register
20 SOUND 1,0                'Kanal A HIGH-Register
30 SOUND 7,&B11111110       'Tonkanal A Freigabe
40 SOUND 8,16               'Kontrolle an Huellkurvenregister 11 & 12
50 SOUND 13,8               'Huellkurvenform
60 INPUT "Geben Sie die Huellkurvenfrequenz ein";F
70 D=1789773/(256*F)
80 L=D AND 255
90 H=D/256
100 SOUND 11,L              'LOW-Register 11 setzen
110 SOUND 12,H              'HIGH-Register 12 setzen
120 GOTO 60
```

In Zeile 50 wird die Hüllkurvenform 8 ausgewählt (betrachten Sie auch Abschnitt 10.1.2). Damit wird eine Tonmudulation erzeugt, die den Werten in den Registern 11 und 12 in den Zeilen 100 und 110 entspricht. Wählen Sie verschiedene Hüllkurvenmuster, die in Abschnitt 10.1.2 gegeben sind, aus und achten Sie auf die Unterschiede.

Jetzt wissen wir, daß der SOUND-Befehl wesentlich schwieriger anzuwenden ist als PLAY. Allerdings können wir mit SOUND direkt auf die PSG-Register zugreifen. Das ermöglicht uns, zum Beispiel einen Laserschuß oder eine Explosion zu simulieren. Diese Möglichkeiten werden wir in Kapitel 12 anwenden.

10.4 BASIC-Wortschatz

Folgende BASIC-Wörter haben wir in diesem Kapitel kennengelernt:

BEEP	10
PLAY	10.1
SOUND	10.3



11

Interaktive Steuerung



11 Interaktive Steuerung

Bis jetzt haben Sie Anweisungsreihen benutzt, die man Programm nennt und die der Computer ausführen muß. Läuft dieses Programm einmal, führt der Computer die Befehle der Reihe nach aus. Allerdings hält das Programm an, wenn eine INPUT-Anweisung verwendet wird und fragt nach einer Eingabe von der Tastatur. Diese Art der Kommunikation mit dem Rechner vermittelt kein Gefühl dafür, wie die Echtzeitsteuerung in einem Aktions-Video-Spiel abläuft. In diesem Kapitel werden wir erklären, wie man MSX-BASIC-Programme mit hoher Geschwindigkeit laufen lassen kann.

11.1 Die INKEY$-Variable und die INPUT$-Funktion

Die einfachste Form der interaktiven Steuerung ist, die Tastatur abzufragen, und wenn eine bestimmte Taste gedrückt wurde, sofort eine entsprechende Aktion durchzuführen. Dabei wird nicht auf RETURN als Eingabeabschluß gewartet. Das kann man erreichen, wenn man INKEY$ oder die Funktion INPUT$ einsetzt.

Das Format für INKEY$ lautet:

```
X$=INKEY$
```

Sie müssen das Ergebnis von INKEY$ in einer Zeichenkettenvariablen ablegen, bevor Sie das Zeichen in einer BASIC-Anweisung verwenden können.

Der zurückgegebene Wert ist ein Nullstring oder eine Zeichenkette, die nur aus einem Zeichen besteht. Eine Null-Zeichenkette (mit der Länge 0) zeigt an, daß kein Zeichen von der Tastatur eingegeben wurde. Ein Einzeichen-String enthält das Zeichen, das auf der Tastatur eingetippt wurde.

Wird INKEY$ verwendet, wird das Zeichen nicht am Bildschirm ausgegeben. Alle Zeichen außer ‹CTRL›+C können im Programm verwendet werden. Mit ‹CTRL›+C wird das Programm abgebrochen.

Betrachten Sie das Beispiel:

```
10 PRINT"Geben Sie J ein,wenn Sie weitermachen, oder N, wenn Sie
aufhoeren wollen."
20 X$=INKEY$                'INKEY$-Variable setzen
30 IF X$="" GOTO 20         'keine Taste gedrueckt
40 IF X$="J" GOTO 100       'Taste J gedrueckt
50 IF X$="N" GOTO 200       'Taste N gedrueckt
60 GOTO 20
100 PRINT "Das Programm wird fortgesetzt..."
200 END
```

Dieses Programm fragt die Tastatur ab und entscheidet dann je nach gedrückter Taste, wohin es springt. Ist die gedrückte Taste ein "J", springt das Programm nach Zeile 100. Ist die Taste "N" gedrückt, wird nach Zeile 200 verzweigt. Die Anweisung IF... GOTO... kennen Sie noch nicht. Wir werden sie im nächsten Kapitel besprechen.

Eine allgemeinere Anweisung, mit der der Computer auf eine menschliche Aktion reagiert, ist INPUT$. Das Format lautet:

```
X$=INPUT$(<n>,[#<Dateinummer>])
```

Die INPUT$-Funktion gibt eine Zeichenkette von <n> Zeichen zurück, die von der Tastatur oder von einer Datei gelesen werden. Wir beschäftigen uns im Moment nur mit der Tastaturoperation. Das obige Programm kann auch so geschrieben werden:

```
10 PRINT"Geben Sie J ein,wenn Sie weitermachen, oder N, wenn Sie
aufhoeren wollen."
20 X$ = INPUT$(1)                'INPUT$-Funktion setzen
30 IF X$ = "J" THEN 100          'Taste J gedrueckt
40 IF X$ = "N" THEN 200 ELSE 20  'Taste J nicht gedrueckt
100 PRINT "Das Programm wird fortgesetzt..."
200 END
```

Wir können nun den Computer so programmieren, daß er in Abhängigkeit von einer gedrückten Taste bestimmte Aktionen durchführt. Das ist die Grundlage für die direkte Echtzeitverarbeitung. MSX-BASIC erlaubt noch viel mehr. Sie können Programme schreiben, die mit dem Joystick, mit einem Paddle oder mit dem Trigger-Eingang zusammenarbeiten. Im nächsten Abschnitt werden wir beschreiben, wie man mit Anweisungen den Programmfluß beeinflussen kann.

11.2 Bedingte Anweisungen

Mit dem bedingten Sprung kann ein Programm abhängig von dem Ergebnis eines Ausdrucks verschiedene Teile durchlaufen. Das allgemeine Format der IF...THEN...ELSE-Anweisung lautet:

```
IF <Ausdruck> THEN <Anweisung>|<Zeilennummer>
   [ELSE <Anweisung>|<Zeilennummer>]


IF <Ausdruck> GOTO <Zeilennummer>
   [ELSE <Anweisung>|<Zeilennummer>]
```

Ist das Ergebnis des Ausdrucks wahr (ungleich 0), wird THEN oder GOTO ausgeführt. Hinter THEN kann entweder ein Ausdruck oder eine Zeilennummer, zu der gesprungen werden soll, stehen. Auf GOTO muß immer eine Zeilennummer folgen. Ist das Ergebnis des Ausdrucks falsch (0), wird THEN oder GOTO übersprungen und der ELSE-Ausdruck (wenn er vorhanden ist) ausgeführt. Sonst wird das Programm mit der nächstmöglichen Anweisung fortgesetzt.

Betrachten Sie nocheinmal die letzten Beispiele. Die IF...THEN...ELSE-Anweisungen können verschachtelt werden. Sie können also innerhalb weiterer IF...THEN...ELSE-Anweisungen eingeschlossen werden. ELSE muß in der gleichen logischen Programmzeile stehen wie der IF...THEN-Ausdruck. Die Verschachtelung ist auf 255 Zeichen begrenzt. Die Anweisung:

```
100 IF A<B THEN PRINT "A ist kleiner al
s B" ELSE IF A>B THEN PRINT "A ist groe
sser als B" ELSE PRINT "A ist gleich B"
```

ist gültig. Innerhalb einer logischen Programmzeile (maximal 255 Zeichen) darf kein RETURN eingegeben werden, obwohl die Zeile größer als eine physikalische Zeile ist (maximal 40 Zeichen). Enthält eine BASIC-Anweisung nicht die gleiche Anzahl von THEN- und ELSE-Ausdrücken, wird jedes ELSE mit dem ihm am nächsten THEN verbunden. Ein Beispiel:

```
100 IF A=B THEN IF B=C THEN PRINT "A=C"
ELSE PRINT "A ungleich C"
```

Ist A=B falsch, führt MSX-BASIC die nächste Zeile aus, weil es keine entsprechende ELSE-Anweisung gibt. Ist A=B wahr und B=C falsch, wird "A ungleich C" ausgegeben, weil die ELSE-Anweisung dem nächsten THEN zugewiesen wird. Ist beides wahr, nämlich A=B und B=C dann gibt das Programm "A=C" aus.

Jetzt wollen wir den Text in unserer Straßenszene mit Hilfe der IF...THEN...ELSE-Anweisung blinken lassen. Laden Sie "SZENE6" in Ihren Computer und fügen Sie die nächsten Zeilen hinzu:

```
593 F=X MOD 32                                        'Blinkanzeiger setzen
595 IF F=0 THEN PSET(185,93),13: PRINT£1,"M S X"  '"MSX" schreiben
597 IF F=20 THEN LINE(185,93)-(230,102),13,BF      '"MSX" loeschen
```

Mit dem MOD-Operator wird der Rest einer Ganzzahldivision einer Variablen zugewiesen. Ein Beispiel:

(11 MOD 5) = 1 (11/5 = 2 Rest 1)
(24 MOD 11) = 2 (24/11 = 2 Rest 2)

Beachten Sie auch, daß in MSX-BASIC die Division durch den Schrägstrich (/) gekennzeichnet wird.

Lassen Sie das Programm laufen und sehen Sie sich an, wie das "M S X"-Zeichen blinkt. Speichern Sie das Programm unter dem Namen "SZENE7" ab.

11.3 Steueranweisungen

Neben der bedingten Anweisung IF...THEN...ELSE enthält das MSX-BASIC verschiedene Sprunganweisungen. Die einfachste Sprunganweisung ohne Bedingung lautet:

```
GOTO <Zeilennummer>
```

Damit wird das Programm veranlaßt, mit der angegebenen <Zeilennummer> weiterzumachen. Diese Anweisung ist gefährlich, weil sie unstrukturiertes Programmieren erlaubt. Unkontrolliertes Anwenden von GOTO-Anweisungen erzeugt unstrukturierte Programme, die man kaum nachvollziehen kann. Es ist viel eleganter, die Programme in Module zu zerlegen und vom Hauptprogramm in die Unterprogramme zu springen. Der Befehl dazu lautet:

```
GOSUB <Zeilennummer>
```

Die Anwendung dieser Anweisung haben wir schon in Kapitel 6 erklärt.

11.3.1 Die Anweisungen ON...GOSUB und ON...GOTO

Im MSX-BASIC ist es möglich, in Abhängigkeit von einem Wert, der in einem Ausdruck übergeben wird, zu verschiedenen Zeilennummern zu springen. Die Anweisungen lauten:

```
ON ‹Ausdruck› GOTO ‹Liste von Zeilennummern›
ON ‹Ausdruck› GOSUB ‹Liste von Zeilennummern›
```

Der Wert von ‹Ausdruck› bestimmt, zu welcher Zeilennummer in der Liste gesprungen werden soll. Ist der Wert keine Ganzzahl, wird der Nachkomma-Anteil gerundet. Ist der Wert 0 oder größer als die Anzahl der angegebenen Listenausdrücke, wird die nächste Anweisung ausgeführt. Ist der Wert negativ oder größer als 255, wird die Meldung "Illegal Function Call" ausgegeben. Ein Beispiel:

```
10 ON X GOTO 100,200,300
```

In Abhängigkeit von dem Wert, den X erhält (1, 2 oder 3), verzweigt das Programm nach Zeile 100, 200 oder 300.

Nachdem Sie nun die Steueranweisungen ON...GOTO kennengelernt haben, wollen wir die direkten Joystickknopf- und Joystick-Steueranweisungen besprechen.

11.3.2 Die Steuerung des Feuerknopfes

Im MSX-BASIC können Sie mit dem Joystick- oder Feuerknopf eines Joysticks den Ablauf eines Video-Spiels steuern.

```
STRIG (‹n›)
STRIG (‹n›) ON/OFF/STOP
ON STRIG GOSUB ‹Liste von Zeilennummern›
```

STRIG (‹n›) gibt den Status des Joystick-Knopfes zurück. Ist der Feuerknopf gedrückt, gibt er -1 (wahr) zurück. Wird er nicht gedrückt, wird der Wert 0 (falsch) zurückgegeben. ‹n› kann im Bereich von 0 bis 4 liegen und bestimmt die Zuordnung des 'Feuerknopfs'. Die Tabelle zeigt die Zuordnung:

‹n›	Joystick-Knopf ist
0	Leertaste
1 oder 3	Joystick-Knopf von Joystick 1
2 oder 4	Joystick-Knopf von Joystick 2

Mit der STRIG(‹n›) ON-Anweisung wird abhängig vom gedrückten Joystick-Knopf eine Aktion durchgeführt wird. Mit ‹n› wird der entsprechende Feuerknopf, wie es vorher beschrieben wurde, ausgewählt. Ist eine Zeilennummer in einer ON STRIG GOSUB-Anweisung angegeben und ist das Programm über den Befehl STRIG (‹n›) ON gekommen, überprüft MSX-BASIC jedesmal nach dem Abarbeiten einer neuen Anweisung, ob der Joystick-Knopf gedrückt wurde. War dies der Fall (wahr = -1), wird der GOSUB-Befehl ausgeführt.

Wird eine STRIG (‹n›) OFF-Anweisung durchlaufen, wird der Joystick-Knopf nicht mehr laufend überprüft.

Nach einem STRIG (‹n›) STOP-Befehl findet keine Abfrage mehr statt. Wird der Joystick-Knopf gedrückt, merkt sich MSX-BASIC das, bis mit STRIG (‹n›) ON die Abfrage wieder zugelassen ist.

ON STRIG GOSUB ‹Liste von Zeilennummern› gibt eine Liste von Zeilen an, auf die gesprungen werden soll, wenn der Joystick-Knopf gedrückt wird. Tritt dieses Ereignis ein, wird automatisch ein STRIG (‹n›) OFF ausgeführt und der Sprung durch GOSUB auf die entsprechende Zeile vorgenommen. Nach der Rückkehr aus einem Unterprogramm, führt BASIC automatisch einen STRIG (‹n›) ON- Befehl aus. Es gibt allerdings eine Ausnahme, nämlich dann, wenn im Unterprogramm ein direkter STRIG (‹n›) OFF-Befehl angegeben wurde.

Probieren Sie das oben Gesagte im folgenden Beispiel aus. (Besitzen Sie die Programmbeispielkassette, können Sie das Programm auch mit LOAD "SCHUSS" von Seite A laden.)

```
1000 'Demonstrationsprogramm fuer ...
1005 STRIG
1070 '
1080 'Hintergrund zeichnen
1090 COLOR 15,4,1: SCREEN 2,3
1100 LINE(0,40)-(255,192),12,BF
1110 LINE(0,177)-(255,192),14,BF
1120 FOR I=0 TO 29
1130   READ C$
1140   LINE(0,I)-(255,I),VAL ("&H"+C$)
1150 NEXT I
1159 'Horizont
1160 DATA 6,6,8,6,8,8,9,9,A,9
1170 DATA 9,A,B,B,A,A,A,B,A,3
1180 DATA 3,2,5,5,7,5,7,7,5,5
1190 '
1200 'Geschoss-sprite definieren
1201 S=1
```

```
1220 GOSUB 1660
1240 '
1250 'Zeilennummer fuer Feuerknopf (Leertaste)
1260 ON STRIG GOSUB 1530
1270 '
1295 'Geschosspositionsvariable (unterer Schirmrand) initialisieren
1298 XB=112:Y=168
1300 '
1310 'Hauptprogrammschleife
1320 IF E=0 THEN E=1: STRIG(0) ON             'Feuerknopf freigeben
1330 IF Y<=10 OR Y=168 THEN Y=168: E=0 ELSE Y=Y-8
1400 PUT SPRITE 1,(XB,Y),8,1                  'Geschoss bewegen
1410 GOTO 1320
1420 '
1530 'Geschoss setzen
1540 STRIG(0) OFF:Y=173:E=1:RETURN
1550 '
1660 'Sprite Routine definieren
1670 S$=""
1680 FOR J=1 TO 32
1690   READ D$
1700   S$=S$+CHR$(VAL("&H"+D$))
1710 NEXT J
1720 SPRITE$(S)= S$
1730 RETURN
1810  'spritedaten des Geschosses
1820 DATA 01,03,07,07,17,1F,1F,1F
1830 DATA 13,00,00,00,00,00,00,00
1840 DATA 00,80,C0,C0,D0,F0,F0,F0
1850 DATA 90,00,00,00,00,00,00,00
```

Dieses Programm ähnelt dem Straßenszenenprogramm, in dem ein Objekt über den Bildschirm bewegt wurde. Es zeichnet eine Abschußrampe und definiert einen Sprite als Geschoß (Bild 11.3.2.1). Der Unterschied besteht darin, daß das Geschoß am unteren Bildschirmrand steht. Drücken Sie die Leertaste, wird es abgeschossen.

Beschäftigen Sie sich mit den Kommentaren, die in diesem Programm enthalten sind, damit Sie alles verstehen. Sind Sie damit zufrieden, speichern Sie das Programm unter dem Namen "SCHUSS" ab. Wir werden es später noch brauchen.

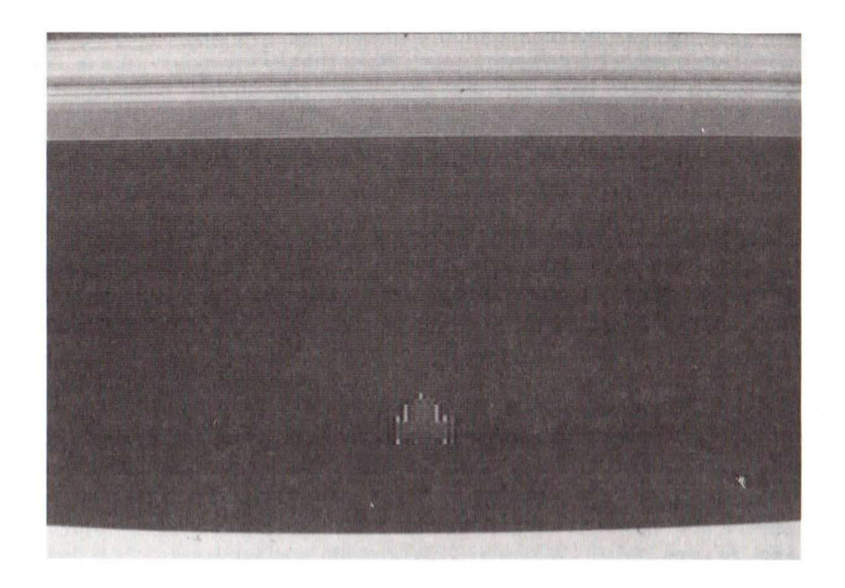


Bild 11.3.2.1 SCHUSS

11.3.3 Die Steuerung des Joysticks

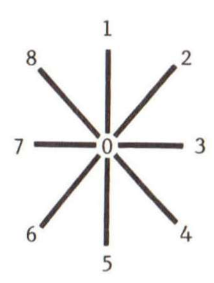
In diesem Abschnitt besprechen wir, wie das Geschoß fliegt. Im MSX-BASIC können Sie die Sprite-Bewegung mit der Anweisung:

```
STICK (<n>)
```

steuern. <n> liegt im Bereich von 0 bis 2 und bestimmt, welcher Joystick bedient wird. Die Tabelle zeigt die Zuordnung:

<n>	Joystick
0	Pfeil- oder Cursortasten
1	Joystick-Port Nummer 1
2	Joystick-Port Nummer 2

STICK (‹n›) gibt die Richtung an, in die der Joystick gebracht wird. Steht er in Neutralstellung, ist der übergebene Wert 0. Sonst gibt er den Wert zurück, der einer Richtung zugeordnet ist:



Jetzt wollen für unser Programm "SCHUSS" ein paar zusätzliche Zeilen schreiben, damit wir den Flug des Geschoßes steuern können. Haben Sie das Programm "SCHUSS" noch im Speicher, geben Sie die untenstehenden Zeilen zusätzlich ein. Andernfalls tippen Sie das Programmsegment ab und laden "SCHUSS" mit MERGE dazu (das Programm-Modul befindet sich unter dem Namen "SCHU2M" auch auf Seite A der Programmbeispielkassette).

```
1003 'STICK
1360 ST=STICK(0)               'Cursorpfeiltasten verwenden
1370 IF ST=3 THEN GOSUB 1430   'Signal rechts
1380 IF ST=7 THEN GOSUB 1480   'Signal links
1430 'Geschoss rechts
1440 XB=XB+2
1450 IF XB>223 THEN XB=223
1460 RETURN
1470 '
1480 'Geschoss links
1490 XB=XB-2
1500 IF XB<0 THEN XB=0
1510 RETURN
1520 '
```

Mit diesen zusätzlichen Zeilen können Sie nun das Geschoß abfeuern und es nach rechts oder links steuern, während es über den Bildschirm fliegt.

Speichern Sie dieses Programm als "SCHUS2" ab. Wir brauchen es später noch.

11.3.4 Sprite-Zusammenstoß

In einem typischen Schießspiel muß das abgefeuerte Geschoß ein Ziel treffen. MSX-BASIC stellt dafür ganz einfache Befehle zur Verfügung.

```
SPRITE ON/OFF/STOP
ON SPRITE GOSUB <Zeilennummer>
```

SPRITE ON/OFF/STOP setzt oder verhindert das Überprüfen der Sprite-Zusammenstöße. Damit die Überprüfung stattfindet, muß eine SPRITE ON-Anweisung im Programm durchlaufen werden. Außerdem muß im ON SPRITE GOSUB-Befehl eine Zeilennummer angegeben sein. MSX-BASIC überprüft jedesmal, wenn eine neue Anweisung durchlaufen wird, ob Sprites miteinander kollidieren. Ist das der Fall, wird in das durch GOSUB angegebene Unterprogramm (Zeilennummer) gesprungen. Sonst wird die nächste Anweisung durchlaufen.

Wird eine SPRITE OFF-Anweisung ausgeführt, wird nicht auf Kollision geprüft.

Läuft das Programm über eine SPRITE STOP-Anweisung, findet keine Überprüfung statt. Das Programm merkt sich aber, daß es Sprite-Kollisionen aufspüren soll. In dem Moment, wo eine SPRITE ON-Anweisung durchlaufen wird, wird diese Überprüfung fortgesetzt.

Mit dem ON SPRITE GOSUB <Zeilennummer>-Kommando wird die Zeilennummer des Unterprogramms mit angegeben, in das gesprungen werden soll, wenn Sprites zusammenstoßen. Findet dieser Vorfall statt, wird automatisch eine SPRITE STOP-Anweisung durchgeführt. Nach der Rückkehr aus dem Unterprogramm (durch RETURN), wird eine SRITE ON-Anweisung durchlaufen. Auch hier gilt als einzige Ausnahme eine direkte SPRITE OFF-Anweisung innerhalb des Unterprogramms.

Jetzt wollen wir weitere Zeilen zum "SCHUS2"-Programm hinzufügen, um die Anwendung der gerade besprochenen Anweisungen zu demonstrieren. Wir definieren ein Sprite, das ein Schiff (das Ziel) darstellt und bewegen es waagrecht. Sie können dann das Geschoß abfeuern und es so steuern, daß es das Ziel trifft. Zusätzlich fügen wir noch Explosionsgeräusche hinzu, um alles etwas aufregender zu machen. Geben Sie das folgende Programmsegment ein. Unter dem Namen "SCHU3M" finden Sie das Modul auch auf der beim Verlag erhältlichen Programmbeispielkassette.

```
1002 'SPRITE ON/OFF,ON SPRITE GOSUB
1200 'Sprites definieren
1210 FOR S=0 TO 2
1220 GOSUB 1660
1230 NEXT S
1231 '
1232 'PSG-Register fuer Explosionsgeraeusch initialisieren
1233 SOUND 7,&B011100              'Ton: Kanal A,B; Geraeusch: Kanal C
1234 SOUND 6,25                    'Geraeuschperiode
1235 SOUND 10,+B 10000             'Huellkurvensteuerung
1236 SOUND 11,0: SOUND 12,50       'Huellkurvenperiode
1240 '
1280 'Kollision ermoeglichen
1290 ON SPRITE GOSUB 1560:SPRITE ON
1293 '
1295 'Variablen initialisieren
1299 XS=-30
1300 '
1310 'Hauptprogrammschleife
1340 IF XS>251 THEN XS=-30 ELSE XS=XS+3
1390 PUT SPRITE 0,(XS,15),1,0            'Schiff sprite
1420 '
1560 'Explosionsroutine
1570 SPRITE OFF
1580 E=0: Y=-39                          'Feuerknopf ruecksetzen
1590 SOUND 13,0                          'Explosionsgeraeusch
1600 PUT SPRITE 0,(0,-39),0,0            'Schiff bewegen
1610 PUT SPRITE 1,((XS+XB)/2,20),15,2    'Explosionssprite
1620 XS=-30:FOR J=1 TO 50: NEXT J        'Schiff ruecksetzen, Zeit
schleife
1630 SPRITE ON
1640 RETURN
1650 '
1750 'Spritedaten
1760 'Schiff
1770 DATA 00,00,00,07,27,27,27,3F
1780 DATA 3A,2F,FF,C0,7F,7F,7F,1F
1790 DATA 00,00,00,80,84,84,E4,E4
1800 DATA B4,FF,07,7E,FE,FE,FE,FC
1860 'Explosion
1870 DATA 03,07,1E,3F,3F,77,6E,CF
1880 DATA SS,F7,B9,7F,6F,1F,3C,07
1890 DATA 80,E0,78,B4,F4,DE,EE,DE
1900 DATA EF,F7,FC,EE,BC,74,F0,C0
```

Alleine läuft dieses Segment nicht. Sie müssen erst "SCHUS2" mit MERGE dazuhängen, um das Programm zu vervollständigen. Haben Sie es zusammengehängt, ist Ihr erstes Videospiel fertig.

Sie sind der Befehlshaber des Zerstörers "Großer Fisch". Es ist Ihre Aufgabe, zu verhindern, daß die feindlichen Schiffe Ihre Flotte einnehmen. Deshalb müssen Sie alle feindlichen Angreifer vernichten. Mit der Leertaste feuern Sie, und mit den Cursortasten steuern Sie Ihr Geschoß nach rechts oder nach links. Viel Glück, Commander (Bild 11.3.4.1)!

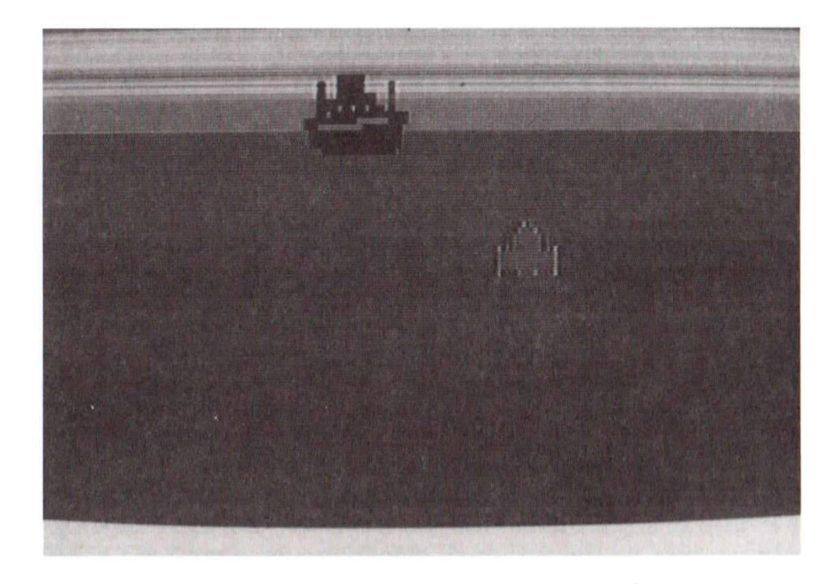


Bild 11.3.4.1 SCHUS3

Beachten Sie, daß wir mit der SOUND-Anweisung die PSG-Tonregister so gesetzt haben, daß sie ein Explosionsgeräusch erzeugen. Die Daten sind in hexadezimaler Darstellung angegeben. Das Programm wurde in Einzelmodulen geschrieben und mit Kommentaren versehen, die die Funktion beschreiben. Sehen Sie sich dieses Beispiel ganz genau an. Falls es notwendig sein sollte, lesen Sie die Abschnitte mit den Anweisungen, die Sie nicht verstanden haben, noch einmal durch, bevor Sie das nächste Kapitel beginnen. Speichern Sie das Programm unter dem Namen "SCHUS3" ab.

11.4 Die RND-Funktion, die Ganzzahldivision und der MOD-Operator

Das einfache Schießspiel aus Abschnitt 11.3 kann leicht gemeistert werden, weil alle Sprites am Bildschirm sich mit der gleichen Geschwindigkeit bewegen. Damit das Video-Spiel kniffeliger wird, setzen wir die MSX-BASIC-Funktion RND(X) ein. Sie erzeugt eine Zufallszahl zwischen 0 und 1.

Wird ein Programm mit RUN gestartet, verwendet RND(X) jedesmal die gleiche Folge von Zufallszahlen, bis der Zufallszahlen-Generator unter Verwendung von X(0) zurückgesetzt wird. Wird X weggelassen oder X›0 benutzt, wird die nächste Zufallszahl generiert. Verwendet man X=0, wird die letzte erzeugte Zahl wiederholt. Ein Beispiel:

```
RND(-TIME)
```

Damit wird der Zufallszahlen-Generator zu Beginn des Programms gesetzt. Jedesmal, wenn mit RUN ein Programm gestartet wird, wird eine unterschiedliche Folge von Zufallszahlen erzeugt, weil -TIME eine negative Zahl ist. Die spezielle Variable TIME ist eine vorzeichenlose Ganzzahl, die den internen Zeitgeber aufruft. Immer, wenn der Video-Display-Prozessor (VDP) einen Interrupt auslöst (60mal pro Sekunde), wird sie automatisch um 1 erhöht.

Erweitern Sie "SCHUS3" um die nächsten Zeilen:

```
1297 I=RND(-TIME)          'Zufallszahlengenerator starten
1340 IF XS>251-SP THEN XS=-30 ELSE XS=XS+3+SP
1350 IF XS=-30 OR XS=-30+SP THEN SP=RND(1)*10
```

Jetzt bewegen sich die feindlichen Schiffe mit einer zufälligen Geschwindigkeit über den Bildschirm. Das Programm "SCHUS3" mit diesen Änderungen befindet sich unter dem Namen "SCHUS4" auch auf der Programmbeispielkassette.

Die RND-Funktion wird sinnvollerweise in Unterprogrammen verwendet, die zufällige Aktionen erzeugen sollen.

11.5 BASIC-Wortschatz

In diesem Kapitel haben wir folgende BASIC-Wörter gelernt:

INKEY$	11.1
INPUT$	11.1
IF...THEN...ELSE	11.2
GOTO	11.3
GOSUB	11.3
ON...GOSUB	11.3.1
ON...GOTO	11.3.1
STRIG (‹n›)	11.3.2
STRIG (‹n›) ON/OFF/STOP	11.3.2
ON STRIG GOSUB	11.3.2
STICK (‹n›)	11.3.3
SPRITE ON/OFF/STOP	11.3.4
ON SPRITE GOSUB	11.3.4
RND(X)	11.4
MOD	11.4

12

Ein vollständiges Spiel: Der Eisplanet



12 Ein vollständiges Spiel: Der Eisplanet

Alle MSX-BASIC-Befehle, die für das Programmieren eines Video-Spiels notwendig sind, haben wir in den vorhergehenden Kapiteln erklärt. Wir werden sie nun verwenden, um ein Computerspiel zu entwickeln. Kommt Ihnen eine Anweisung in diesem Kapitel unbekannt vor, frischen Sie Ihr Gedächtnis wieder auf, indem Sie die vorhergehenden Abschnitte lesen. Wir werden in diesem Kapitel die BASIC-Wörter nicht erklären, sondern uns auf den Entwurf des Spiels konzentrieren.

Wir führen den Entwurf von Anfang bis Ende durch. Die notwendigen Schritte in einem TOP-DOWN-Entwurf eines Video-Spiels sind:

1. Entwickeln Sie eine Spielidee und -- eine Geschichte.
2. Rufen Sie sich die Steuermodi ins Gedächtnis.
3. Entwerfen Sie Hintergrund und Figuren.
4. Denken Sie über Musik und Geräuscheffekte nach.
5. Programmieren Sie.

Die ersten drei Schritte stehen miteinander in Beziehung und hängen von Ihrer Kreativität ab. Damit Ihr Spiel interessant wird, sollten Sie ein gutes Manuskript haben und einen spektakulären Bildschirm-Hintergrund und gut durchdachte Bewegungsabläufe programmieren.

Beim Programmieren sind vier wichtige Komponenten zu beachten. Wir haben sie zusammengefaßt in Kapitel 11 dargestellt. Sie lauten:

1. Initialisierung
2. Bedingungsüberprüfung
3. Aktion
4. Bildschirmanzeige.

Sehen wir uns nun die einzelnen Punkte an.

12.1 Das Spiel

Das Fliegen in fremder Umgebung hat die Menschen schon immer fasziniert. Wir entwerfen ein Spiel, bei dem ein Raumkrieger durch einen Eissturm von seiner Einheit getrennt wird.

Damit er die Zivilisation wiederfindet, muß der Held dem Sturm trotzen. Der Himmel ist voller Eisblöcke, die aus seltsamen Materialien eines Planeten bestehen. Die Eisblöcke werden vom Raumschiff angezogen und scheinen ihm zu folgen. Um eine Kollision zu vermeiden, muß unser Held auf Kurs der Eisblöcke bleiben und darf erst im letzten Moment seine Höhe ändern, damit er der "künstlichen Intelligenz" der Eisblöcke nicht unterliegt.

Nach dem Eissturm muß unser Held das beschädigte Raumschiff durch eine Reihe von Nachtankstationen manövrieren, um zur Reparaturstation zu gelangen. Ein magnetischer Sturm rüttelt das Raumschiff durch und macht das Andocken sehr schwierig. Das automatische Radar und der Magnetflußanzeiger arbeiten nur in der Nähe der Nachtankstationen. Haben Sie einmal Kontakt, geleiten Sie das Raumschiff zur Nachtankstation. Während des Durchfliegens des magnetischen Sturms wird sehr viel Treibstoff verbraucht. Unser Held muß damit sorgsam umgehen und darf keine Chance auslassen, nachzutanken.

Nach langem und hartem Einsatz fällt auch der letzte Ausrüstungsgegenstand aus. Das Radar und der Magnetflußanzeiger sind nicht mehr in Betrieb. Das einfache, aber sehr interessante Spiel beginnt von neuem.

12.2 Wie man das Spiel steuert

In diesem Spiel gibt es zwei Ebenen, in denen das Raumschiff mit den vier Pfeil-Tasten (Cursor-Bewegungstasten) gesteuert wird:

^ steigen
v fallen
› rechts
‹ links

Damit das Programmieren nicht gar so schwer wird, bleibt die Geschwindigkeit des Raumschiffs konstant.

12.3 Bildschirmdarstellung

Die Eisblöcke und die Auftank- und Reparaturstationen sind auf der linken Seite des Bildschirms zu sehen. Auf der rechten Seite zeigen wir den Radarschirm, die Treibstoffmenge und andere Flugdaten. Die endgültige Anzeige sieht aus wie in den Bildern 12.3.1 und 12.3.2 gezeigt.

Um die verschiedenen Objekte am Bildschirm darzustellen, müssen wir Sprite-Muster definieren. Nehmen Sie ein in 8 x 8 Kästchen unterteiltes Stück Papier, zeichnen Sie jedes Sprite auf und übersetzen Sie den Code dann in das hexadezimale System. Da DATA- Anweisungen an beliebiger Stelle im Programm placiert werden können, speichern wir die Sprite-Muster in DATA-Anweisungen mit Zeilennummern oberhalb von 7840 ab. Wir nehmen deshalb höhere Zeilennummern, damit Sie auch ganz sicher am Ende des Programms stehen. Die Tabellen zeigen die Sprites:

```
7860 'Muster 0: Raumschiffrumpf
7880 DATA 02,02,07,07,0F,0F,2F,3F
7900 DATA 3F,3F,3E,07,03,00,00,00
7920 DATA 40,40,E0,E0,F0,F0,F4,EC
7940 DATA DC,DC,3C,E0,C0,00,00,00

7960 'Muster 1: Positionslichter des Schiffes
7980 DATA 02,02,04,04,09,0A,2A,2A
8000 DATA 2A,20,20,00,00,00,00,00
8020 DATA 40,00,00,00,80,00,00,00
8040 DATA 00,00,00,00,00,00,00,00


8060 'Muster 2: Auftankstation
8080 DATA 3F,7F,FF,FF,FF,FC,F8,F8
8100 DATA F8,F8,DC,EF,DE,DF,47,07
8120 DATA FC,FE,FF,FF,FF,3F,16,16
8140 DATA 16,16,37,EF,1F,FF,FE,FC
```

```
8160 'Muster 3: Schiff am Radarschirm
8180 DATA 00,00,00,00,02,02,07,0C
8200 DATA 78,1C,0F,00,00,00,00,00
8220 DATA 00,00,00,00,40,40,E0,30
8240 DATA 1E,38,F0,00,00,00,00,00

8260 'Muster 4: Pfeil nach unten
8280 DATA 00,00,00,00,00,00,00,00
8300 DATA 38,38,38,38,FE,7C,38,10
8320 DATA 00,00,00,00,00,00,00,00
8340 DATA 00,00,00,00,00,00,00,00

8360 'Muster 5: Pfeil nach rechts
8380 DATA 00,00,00,00,00,00,00,00
8400 DATA 08,0C,FE,FF,FE,0C,08,00
8420 DATA 00,00,00,00,00,00,00,00
8440 DATA 00,00,00,00,00,00,00,00

8460 'Muster 6: Pfeil nach oben
8480 DATA 00,00,00,00,00,00,00,00
8500 DATA 10,38,7C,FE,38,38,38,38
8520 DATA 00,00,00,00,00,00,00,00
8540 DATA 00,00,00,00,00,00,00,00

8560 'Muster 7: Pfeil nach links
8580 DATA 00,00,00,00,00,00,00,00
8600 DATA 10,30,7F,FF,7F,30,10,00
8620 DATA 00,00,00,00,00,00,00,00
8640 DATA 00,00,00,00,00,00,00,00

8660 'Muster 8: Eisblock
8680 DATA 00,00,01,07,1F,3F,3B,3D
8700 DATA 3F,2F,0F,1D,0F,0D,00,00
8720 DATA 00,00,C0,F8,FC,FC,B8,DC
8740 DATA FC,7C,F8,F8,F0,F0,E0,00

8760 'Muster 9: Explosion
8780 DATA 00,03,00,08,03,19,03,28
8800 DATA 07,03,26,2C,00,09,00,00
8820 DATA 00,04,60,00,2C,80,00,26
8840 DATA 80,6C,C0,10,48,00,00,00
```

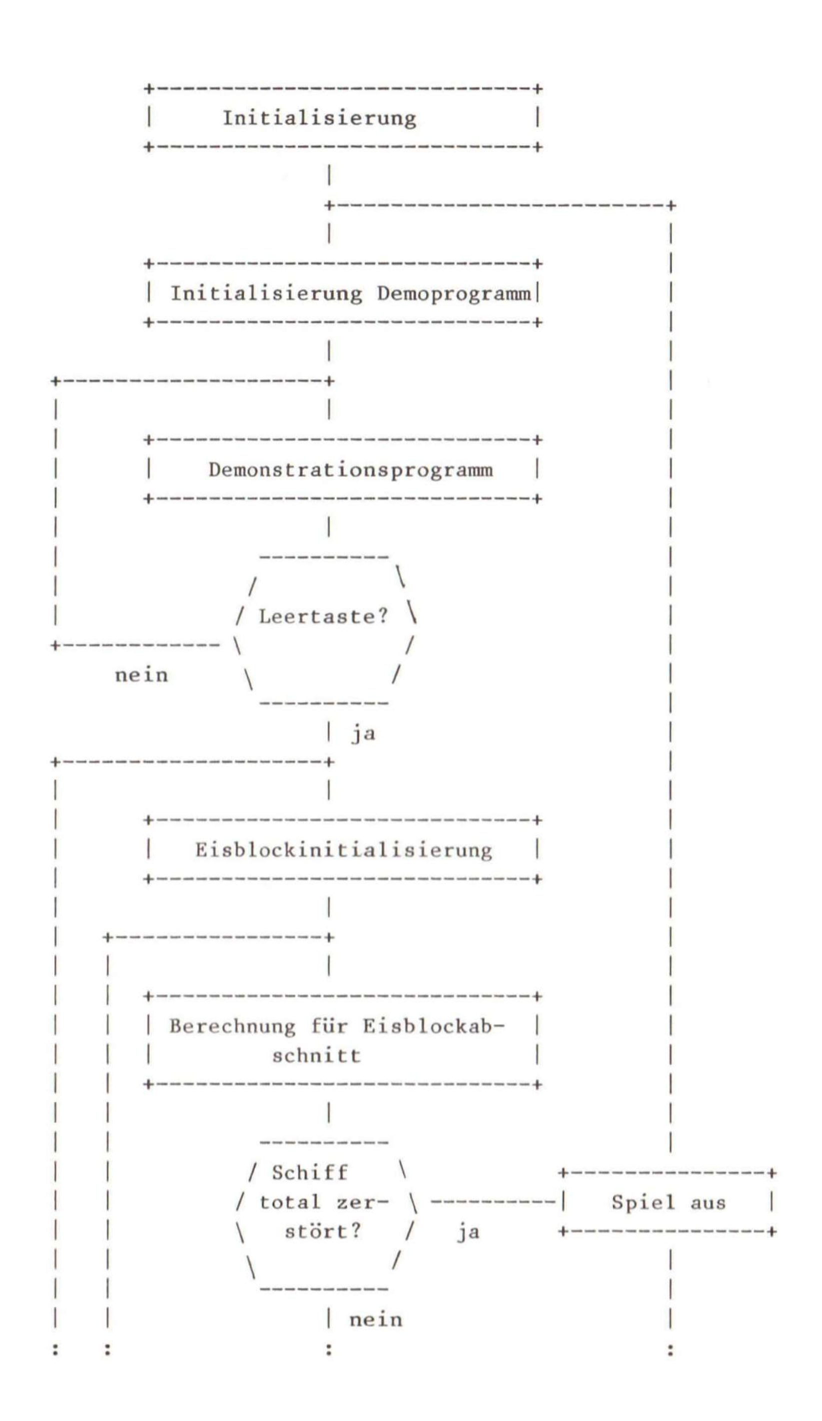
Jetzt haben wir die Sprite-Muster entworfen, die wir in unserem Video-Spiel einsetzen. Wir verwenden für das Raumschiff und seinen Schatten das gleiche Sprite. Ebenso für die Auftankstationen und deren Schatten.

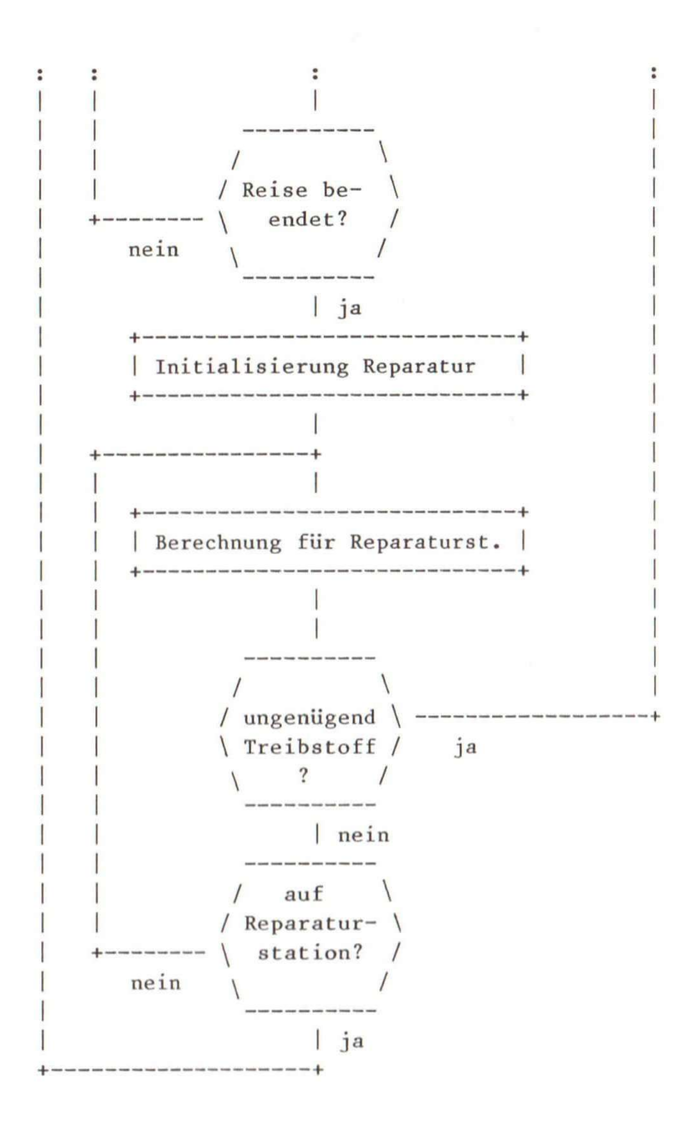
12.4 Die Einzelheiten der Programmierung

Wir beginnen mit der Programmierung, indem wir die Einzelteile des Hauptprogramms aufschreiben:

1. Initialisierung
2. Demonstrationsprogramm - Selbstdemonstration
3. Eisblockanwahl - das Fliegen durch den Eissturm
4. Der Reparaturabschnitt - Das Fliegen über die Auftankstationen zur Reparaturstation.

Nach diesem Entwurf können wir ein Flußdiagramm zeichnen, das uns bei der Programmierung hilft:


Initialisierung
Initialisierung Demoprogramm
Demonstrationsprogramm
Leertaste?
nein
ja
Eisblockinitialisierung
Berechnung für Eisblockab-
schnitt
Schiff
total zer-
stört?
ja
Spiel aus
nein



Im allgemeinen sollte Ihr Flußdiagramm nur die Hauptverzweigungen enthalten, weil es sonst nur unnötig kompliziert wird, wenn Sie jede einzelne Verzweigung des Programms darin aufnehmen.

Um diese Schwierigkeit zu umgehen, benutzen wir eine Pseudocode- Schreibweise und bezeichnen über Zeilennummern Blöcke, die die Module bilden. Jedes Modul sollte eine Funktion durchführen, die in Ihrem Programm immer wieder benutzt wird. Vergewissern Sie sich, daß Ihnen genügend Platz zwischen den Zeilennummern bleibt, damit Sie

neue Zeilen hinzufügen können, wenn Ihnen noch etwas Wichtiges einfällt. Ein typischer Entwurf könnte so aussehen:

Zeilennummer	**Funktion**
1000	Programmstart
1360	Demonstrationsabschnitt
1600	Eisblockabschnitt
1880	Reparaturabschnitt
2340	Demonstrationsunterprogramm
3240	Joystick-Knopf-Bedienroutine
3320	Gebilde zeichnen
3600	Eisblockunterprogramm
4300	Reparaturunterprogramm
5660	Punktanzahl anzeigen
5780	Magnetfluß nach unten
5800	Steuerung - Raumschiff nach unten
5840	Steuerung - Raumschiff nach rechts unten
5860	Magnetfluß nach rechts
5880	Steuerung - Raumschiff nach rechts
5920	Steuerung - Raumschiff nach rechts oben
5940	Magnetfluß nach oben
5960	Steuerung - Raumschiff nach oben
6000	Steuerung - Raumschiff nach links oben
6020	Magnetfluß nach links
6040	Steuerung - Raumschiff nach links
6080	Steuerung - Raumschiff nach links unten
6220	Eine neue Eisblockroutine erzeugen
6420	Explosionseffekt-Unterprogramm
6660	Sprite-Priorität schalten
6760	Hintergrund und Instrumentierpanel zeichnen
7300	Sprite-Muster definieren
7560	Verzögerungs-Unterprogramm
7620	Lautstärke und Tonhöhe des PSG-Kanals init.
7720	PSG-Kanal schließen

Die Zeilennummer gibt die erste Zeile des Moduls mit der Funktion an. Wir schreiben das Programm mit Hilfe dieser Pseudocode-Tabelle. Anschließend finden Sie eine Liste der Variablen, die in diesem Programm verwendet werden:

Variablen-Name	Beschreibung
AC	Farbe des Sprite-Pfeils
CN	Marke für die aktuelle Treibstoffanzeige
CU	Zähler für das Landen und die Verzögerung, wenn die Reparaturstation erreicht ist
DD	Ablesen des Entfernungsmessers
DH	Schrittgröße der waagrechten Bewegung des Raumschiffs
DM	Grad der Raumschiffbeschädigung
DT	gereiste Entfernung
DV	senkrechte Bewegung des Raumschiffs
DL	niedrigste Höhe des Raumschiffs
FC	Farbe des Füllstandsanzeigers
FD	Marke, ob Treibstoff ansteigt oder weggenommen wird
FL	Resttreibstoff
FM	Rate des Treibstoffverbrauchs
FR	Menge des überschüssigen Treibstoffs
HP	Höhe des Raumschiffs
HS	Höhe der Nachtankstation
I	Zwischenzähler
IN	Zahl der Eisblöcke
J	Zwischenzähler
LC	Farbe der Punkte am Hintergrund, die die Raumschiffbewegung zeigen
LL	linke Grenze des Raumschiffs
LV	Höhenzähler
NI	Zahl des Soll-Eisblockzählers
PC	Farbe des Raumschiffs auf dem Radar
PI	= 3.141592654
PO	Marke, die die Sprite-Prioritätsänderung anzeigt
P1	Positionslichter des Raumschiffs
P2	Raumschiffrumpf
R3	Farbe des Raumschiffs, die am Radar angezeigt wird, wenn das Raumschiff nicht auf gleicher Höhe mit der Auftankstation ist
R2	Farbe des Raumschiffs, das am Radar angezeigt wird, wenn das Raumschiff mit der Auftankstation übereinstimmt
RC	Farbe des Sprites am Radar
RL	rechte Grenze des Raumschiffs
SC	Punktezahl
S1	Spitze der Auftankstation
S2	Basis der Auftankstation

Variablen-Name	Beschreibung
UL	obere Grenze des Raumschiffs
WC	Farbe der Pfeile
WH	Sprite-Name für den waagrechten Pfeil
WV	Sprite-Name für den senkrechten Pfeil
XP	horizontale Position des Raumschiffs am Bildschirm
XS	horizontale Position der Auftankstation am Bildschirm
Y	vertikale Position des Raumschiffs am Bildschirm
YP	vertikale Position der Auftankstation am Bildschirm

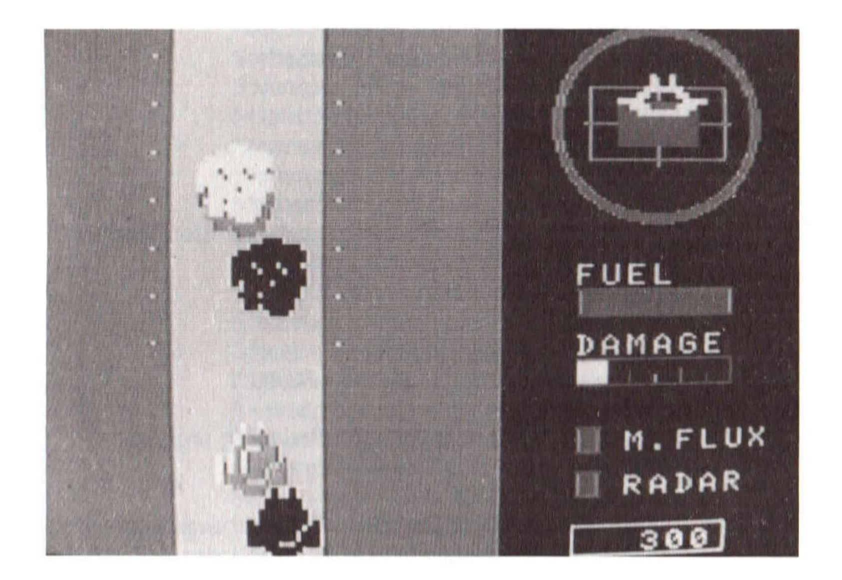


Bild 12.3.1 Programm PLANET

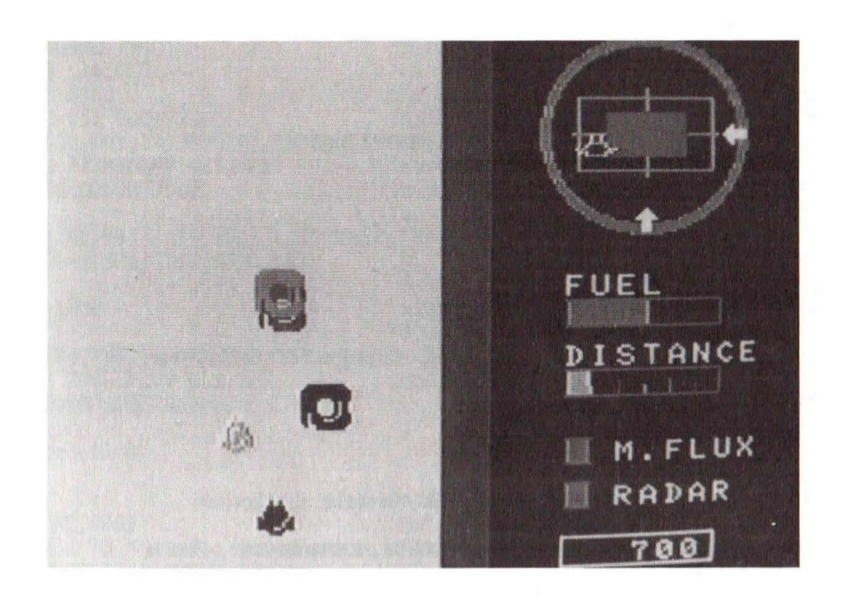


Bild 12.3.2 Programm PLANET

12.5 Das Programmlisting

Auf den nächsten Seiten finden Sie das vollständige Programm. Innerhalb des Programms finden Sie viele Kommentarzeilen, die Ihnen die Funktion der einzelnen Module oder Zeilen erklären. Versuchen Sie zusammen mit der Pseudocode-Tabelle die Funktionsmodule herauszusuchen, damit Sie einen Überblick über die Programmstruktur bekommen. Studieren Sie dann jedes Modul, damit Sie die Einzelheiten verstehen. Sie können das Programm ohne die Kommentaranweisungen eingeben. Haben Sie die Programmbeispielkassette vorliegen, können Sie es mit CLOAD laden. Beachten Sie, daß es zwei Versionen dieses Programms gibt. Das eine wird mit "PLNT32" und das andere mit "PLNT16" bezeichnet. Hat Ihr Computer nur 16K Byte RAM, müssen Sie CLOAD "PLNT16" benutzen. Sonst können Sie mit CLOAD "PLNT32" arbeiten. Beide Versionen sind mit Ausnahme der Kommentaranweisungen identisch.

```
1000 '******************
1002 '*     PLANET     *
1004 '******************
1020 '
1040 DEFINT A-Z
1060 A=RND(-TIME):PI#=3.141592654#
1080                                 'Musikstrings
1036 C1$="T255L5004CDEFGAB05CDEFGAB06C"
1120 C2$="T255L5003CDEFGAB04CDEFGAB05C"
1140 E1$="V13T40L1004DAFD"
1160 E2$="V11T40L1004FDAF"
1180 G1$="V15T40L2004CGL40EGFEL805C"
1200 G2$="V13T40L2003GECDL804G"
1220 P1$="T255L5006C05BAGFEDC04BAGFEDC"
1240 P2$="T255L5005C04BAGFEDC03BAGFEDC"
1260 SOUND 7,&B011100            'PSG Kanaele initialisieren
1280 '
1300 '"GRP:" ist Bildschirmausgabekanal
1320 OPEN"GRP:"AS#1
1340 '
1360 'Demoabschnitt
1380 GOSUB 7720                  'PSG Kanaele schliessen
1400 CLS
1420 SC=0:LV=0:HS=25             'Punkte,Hoehenmesser, Hoehe
1440 COLOR 15,4,1:SCREEN 2,3     'dunkelblauer Hintergrund
1460 PSET(72,90),14:PRINT#1,"BITTE WARTEN..."
1480 PLAY G1$,G2$
1500 RESTORE 7880                'Datenzeiger setzen
1520 N=7:GOSUB 7300              'Spritemuster definieren
1540 COLOR ,14:CLS               'grauer Hintergrund
1560 GOSUB 2340                  'Demoprogramm ausfuehren
1580 '
1600 'Eisblock Abschnitt
1620 GOSUB 7720                  'PSG Kanaele schliessen
1640 COLOR ,13:SCREEN 2,3        'violetter Hintergrund
1660 PSET(80,88),13:PRINT#1,"SPIELER BEREIT"
1680 PLAY P1$,P2$                'Musik Spieler bereit
1700 RESTORE 7880                'Datenzeiger setzen
1720 N=9:GOSUB 7300              'Spritemuster definieren
1740 COLOR ,14:CLS               'grauer Hintergrund
1760 GOSUB 3600                  'Eisblockabschnitt
1780 IF DM=48 THEN 2260          'Schiff beschaedigt
1800 PSET(12,85),15:PRINT#1,"ZUM HANGAR"
1820 GOSUB 7720:PLAY C1$,C2$     'Musik wechseln
1840 N=2000:GOSUB 7560           'kurze Verzoegerungsschleife
1860 '
1880 'Hangar (Reparaturstation)
1900 GOSUB 7720                  'PSG Kanaele schliessen
1920 COLOR ,12:SCREEN 2,2        'dunkelgruener Hintergrund
1940 PSET(80,88),13:PRINT#1,"SPIELER BEREIT"
```

```
1960 PLAY P1$,P2$                       'Musik Spieler bereit
1980 RESTORE 7880                       'Datenzeiger setzen
2000 N=7:GOSUB 7300                     'Spritemuster
2020 COLOR ,14:CLS                      'Hintergrund grau
2040 GOSUB 4360                         'Reparaturstation
2060 IF DD<231 THEN 2260                'kann Hangar nicht erreichen
2080 PSET(30,80),14:PRINT#1,"SCHIFF REPARIERT"
2100 GOSUB 7720:PLAY C1$,C2$            'Musik wechseln
2120 N=2000:GOSUB 7560                  'kurze Verzoegerung
2140                                    'zu restlichem Treibstoff dazu
2160 LINE(185,96)-(231,103),1,BF:SC=SC+FR*10:GOSUB 5660
2180 N=5000:GOSUB 7560:GOTO 1600
2200 '
2220 GOTO 1900                          'wieder zum Eisblockabschnitt
2240                                    'Spel zu Ende
2260 PSET(46,85),14:PRINT#1,"SPIELENDE":GOSUB 5660
2280 GOSUB 7720:PLAY E1$,E2$            'Musik fuer Spielende
2300 N=5000:GOSUB 7560:GOTO 1360
2320 '
2340 'Demoroutine
2360 '
2380                                    'sphaerische Gebilde zeichnen
2400 RESTORE 8900                       'Datenzeiger setzen
2420 FOR I=1 TO 5
2440  READ CX,CY,R:GOSUB 3320
2460 NEXT I
2480                                    'andere Gebilde zeichnen
2500 DRAW"BM83,0C1D165G11D15R3U12E11U30H2U138"
2520 PAINT(73,190),1
2540 LINE(8,0)-(8,79),15
2560 DRAW"BM8,80C1F16R32E16U80R3D84G16L31H5"
2580 PAINT(73,1),1
2600 DRAW"BM85,137C1R72F18R80U1L81H18L72"
2620 DRAW"BM224,24C1F11R21U2L22F1R22"
2640 LINE(224,0)-(224,23),15
2660 DRAW"BM192,191C1U16E8R32F8H4L32G8D12"
2680 PAINT(193,191),1
2700 LINE(240,176)-(240,191),15
2720 '
2740 GOSUB 7620                         'PSG Kanal initialisieren
2760 SOUND 6,15:SOUND 10,12             'Geraeuschperiode initialisieren
2780 '
2800 'Hauptschleife fuer Demoroutine
2820 ON STRIG GOSUB 3240:STRIG(0) ON
2840 PSET(84,180),14:PRINT#1,"LEERTASTE DRUECKEN"
2860 FOR XP=88 TO 128 STEP 20           'waagrechte Position
2880  FOR HP=5 TO 45 STEP 20            'Hoehe
2900   GOSUB 6660                       'Prioritaet fuer Sprites aendern
2920                                    'Maschinenlaerm
2940   SOUND 0,100-HP*1.5:SOUND 6,30-HP/2:SOUND 10,10+HP/20
```

```
2960                                'Sprites neu setzen
2980    PUT SPRITE S1,(104,7),3,2
3000    PUT SPRITE S2,(106,18),12,2
3020    PUT SPRITE 4,(134,50),1,2
3040                                'Sprite und Schatten des Raumschiffs bewegen
3060    FOR YP=193 TO -20 STEP -2
3080     PUT SPRITE P1,(XP,YP),15,1
3100     PUT SPRITE P2,(XP,YP),9,0
3120     PUT SPRITE 5,(XP+HP,YP+12+HP/2),1,0
3140    NEXT YP
3160   NEXT HP
3180 NEXT XP
3200 GOTO 2860
3220 '
3240 'Aktionstastenbedienroutine
3260 STRIG(0) OFF:RETURN 3280
3280 RETURN
3300 '
3320 'Routine fuer sphaerische Gebilde
3340 S=(R+1)/4
3360 CIRCLE(CX,CY),R,1
3380                                'Schatten
3400 CIRCLE(CX+S,CY+S),R,1
3420 PAINT(CX+S*2,CY+S*4),1
3440                                'Positionslichter zeichnen
3460 CIRCLE(CX+S,CY+S),R,14
3480 CIRCLE(CX,CY),R,15,PI#/3,PI#*5/4
3500 CIRCLE(CX,CY),S*2+1,15,PI#*3/4,PI#
3520 CIRCLE(CX,CY),S*2,15,PI#*3/4,PI#
3540 CIRCLE(CX,CY),S*2-1,15,PI#*3/4,PI#
3560 RETURN
3580 '
3600 'Eisblockroutine
3620 '
3640                                'Hintergrund und Panel zeichnen
3660 GOSUB 6760:PSET(184,111),1:PRINT #1,"TREFFER"
3680 LINE(0,0)-(55,192),13,BF:LINE(104,0)-(159,192),13,BF
3700 GOSUB 5660                     'Punkte anzeigen
3720                                'Variablen initialisieren
3740 LV=LV+1:NI=LV*5+20:IN=0:LC=15
3760 P1=1:P2=2:S1=0:S2=3
3780 Y=-31:R1=15:R2=9:DM=0
3800 XS=80:HS=20:XP=50:HP=10
3820 DV=3:DH=4:DL=1:RL=80:UL=20:LL=48
3840 '
3860 'Hauptschleife fuer Eisblockabschnitt
3880    IF Y>207 THEN SC=SC+100:GOSUB 5660:Y=-31:GOSUB 6220
3900    Y=Y+16:PSET(50,Y+32),LC:PSET(109,Y+32),LC
3920                                'Ueberpruefung Kollision
```

```
3940    IF P1<>1 OR XS-XP>19 OR XS-XP<-19 OR Y-160>17 OR Y-160<-29 THEN 3980
3960    GOSUB 6420:GOSUB 6220           'Kollision
3980    IF DM=48 THEN RETURN           'Raumschiff voellig zerstoert
4000                                   'Cursorsteuerung
4020    ST=STICK(0):ON ST GOSUB 5800,5840,5880,5920,5960,6000,6040,6080
4040                                   'Maschinngeraeusch erzeugen
4060    SOUND 0,100-HP*1.5:SOUND 6,30-HP/2:SOUND 10,10+HP/20
4080                                   'Eisblock auf Schiff zubewegen
4100    IF HS<>HP+2 THEN HS=HS-RS*(2*(HS-3<HP)+1)
4120    IF XS<>XP THEN XS=XS-2*(XS<XP)-1
4140    GOSUB 6660                     'Spriteprioritaet aendern
4160                                   'Sprites neu setzen
4180    PUT SPRITE P1,(XP,160-HP),15,1
4200    PUT SPRITE P2,(XP,160-HP),9,0
4220    PUT SPRITE 4,(XP+HP/2,163),1,0
4240    PUT SPRITE S1,(XS,Y-HS),15,8
4260    PUT SPRITE S2,(XS+1,Y-HS+5),7,8
4280    PUT SPRITE 5,(XS+HS/2,Y+8),1,8
4300    PUT SPRITE 6,(129+XP,35-HP),15,3
4320    GOTO 3880
4340 '
4360 'Reparaturroutine
4380 '
4400                                   'Variablen initialisieren
4420 Y=0:WC=15:R1=15:R2=9
4440 XP=70:HP=10
4460 DV=1:DH=1:DL=0:RL=144:UL=50:LL=0
4480 FL=231:DT=0:CN=0:CU=0:AP=0
4500 GOSUB 6120                        'Hangarposition erzeugen
4520                                   'Hintergrund und Panel zeichnen
4540 LINE(0,0)-(15,192),12,BF
4560 LINE(144,0)-(159,192),12,BF
4580 GOSUB 6760:PSET(184,111),1:PRINT #1,"ENTFERN."
4600 GOSUB 5660                        'Punktestand anzeigen
4620 GOSUB 7620                        'PSG initialisieren
4640 '
4660 'Hauptschleife fuer Reparaturroutine
4680    IF CU>0 AND HP=0 THEN RETURN
4700    SC=SC+1
4720    DT=DT+1:DD=184+DT/8
4740                                   'entfernungsabhaengige Aktionen
4760    IF DD>221 THEN AP=AP+2:PSET(27,AP),15:PSET(132,AP),15
4780    IF DD>231 THEN DD=231:FR=FL-184:FL=184:FC=3
4800    LINE(DD,120)-(DD,127),11
4820    IF DD=208 THEN WC=0:LINE(184,144)-(191,151),8,BF:SOUND 2,40:SOUND 13,0
4840    IF DD=216 THEN R1=0:R2=0:LINE(184,160)-(191,167),8,BF:SOUND 2,40:SOUND 13,0
4860    IF Y>207 THEN Y=0:GOSUB 5660:GOSUB 6120
4880    Y=Y+2:FD=-1:FC=1:FM=8-DT/90:AC=WC
4900    IF POINT(208-(XS-XP),36-(HP-HS))<>4 OR Y-160>17 OR Y-160<-19 THEN 4980
```

```
4920                                    'Raumschiffsteuerung durch Auftankstation
4940     AC=0:IF P1=1 THEN FD=1:FC=3:FM=1:SC=SC+19:GOSUB 5660:GOTO 5160 ELSE 5160
4960                                    'magnetischer Flussanzeiger
4980     IF TIME>16383 THEN TIME=0
5000     IF FL<185 AND HP=0 THEN GOSUB 5660:GOTO 5060
5020     XD=((TIME MOD 131)>61)+2:ON XD GOSUB 5860,6020
5040     HD=((TIME MOD 250)>125)+2:ON HD GOSUB 5780,5940
5060     IF FL<185 THEN FC=3:CU=CU+10:N=CU:GOSUB 7560:GOSUB 5780:GOTO 5160
5080                                    'Cursorsteuerung
5100     ST=STICK(0):ON ST GOSUB 5800,5840,5880,5920,5960,6000,6040,6080
5120     IF ST<>0 THEN FM=2
5140                                    'Treibstoffanzeige neu setzen
5160     IF FL>231 THEN FL=231
5180     IF FL<184 THEN FL=184
5200     LINE(FL,96)-(FL,103),FC
5220     CN=(CN MOD FM)+1
5240     IF CN=1 THEN FL=FL+FD
5260                                    'Maschinengeraeusch
5280     SOUND 0,100-HP*1.5:SOUND 6,30-HP/2:SOUND 10,10+HP/20
5300     GOSUB 6660                     'Spriteprioritaet aendern
5320     IF P1<>P0 THEN SOUND 2,80:SOUND 13,0
5340     P0=P1
5360     IF HP>HS-12 AND HP<HS+7 AND XS-XP<20 AND XS-XP>-20 THEN PC=RC ELSE PC=0
5380                                    'Sprites neu setzen
5400     PUT SPRITE P1,(XP,160-HP),15,1
5420     PUT SPRITE P2,(XP,160-HP),9,0
5440     PUT SPRITE 4,(XP+HP/2,163),1,0
5460     PUT SPRITE S1,(XS,Y-HS),3,2
5480     PUT SPRITE S2,(XS+1,Y-HS+5),12,2
5500     PUT SPRITE 5,(XS+HS/2,Y+8),1,2
5520     PUT SPRITE 6,(200-(XS-XP),28-(HP-HS)*2),PC,3
5540     PUT SPRITE 7,(WX,28),AC,WH
5560     PUT SPRITE 8,(205,WY),AC,WV
5580     GOTO 4680
5600 '
5620 'allgemeine Routinen
5640 '
5660 'Punktestand anzeigen
5680 LINE(184,180)-(231,189),1,BF
5700 PSET(190,182),1:PRINT#1,USING"#####";SC
5720 RETURN
5740 '
5760 'Richtungsteuerroutinen
5780 WV=4:WY=0                          'Fluss nach unten
5800 HP=HP-DV:IF HP<DL THEN HP=DL       'Schiff nach unten
5820 RETURN
5840 GOSUB 5780:GOSUB 5860:RETURN       'Schiff nach rechts unten
5860 WH=5:WX=176                        'Fluss nach rechts
5880 XP=XP+DH:IF XP>RL THEN XP=RL       'Schiff nach rechts
5900 RETURN
```

```
5920 GOSUB 5940:GOSUB 5860:RETURN      'Schiff nach rechts oben
5940 WV=6:WY=56                        'Fluss nach oben
5960 HP=HP+DV:IF HP>UL THEN HP=UL      'Schiff nach oben
5980 RETURN
6000 GOSUB 5940:GOSUB 6020:RETURN      'Schiff nach links obn
6020 WH=7:WX=233                       'Fluss nach links
6040 XP=XP-DV:IF XP<LL THEN XP=LL      'Schiff nach links
6060 RETURN
6080 GOSUB 6020:GOSUB 5780:RETURN      'Schiff nach links unten
6100 '
6120 'Position der Reparaturstation
6140 HS=INT(RND(1)*35)+10
6160 XS=INT(RND(1)*64)+40
6180 RETURN
6200 '
6220 'Eisblockposition
6240 IN=IN+1:IF IN=NI THEN RETURN 6380
6260 PUT SPRITE S1,(0,207),0,3
6280 PUT SPRITE S2,(0,207),0,3
6300 HS=INT(RND(1)*10)+9
6320 XS=INT(RND(1)*48)+40
6340 RS=INT(RND(1)*1.4)+2
6360 IF LC=15 THEN LC=13 ELSE LC=15
6380 RETURN
6400 '
6420 'Explosion
6440 SOUND 10,&B10000:SOUND 13,0
6460 PUT SPRITE S2,(0,207),0,0
6480 FOR I=Y-HS TO Y-HS+40 STEP 4
6500  PUT SPRITE 0,(XS,I),15,9
6520  PUT SPRITE 5,(XS+HS/2,I+8),1,9
6540  PUT SPRITE P2,(XP,160-HP),7,0
6560 NEXT I
6580 PUT SPRITE 0,(0,207),0,9:Y=-31
6600 DM=DM+2:LINE(182+DM,120)-(182+DM,127):LINE(183+DM,120)-(183+DM,127)
6620 RETURN
6640 '
6660 'Spriteprioritaet aendern
6680 IF HP>HS THEN P1=0:P2=1:S1=2:S2=3:RC=R1:RETURN
6700 IF HP>HS-5 THEN P1=1:P2=2:S1=0:S2=3:RC=R2:RETURN
6720 P1=2:P2=3:S1=0:S2=1:RC=R1:RETURN
6740 '
6760 'Hintergrund und Panel zeichnen
6780 LINE(160,0)-(256,192),1,BF
6800 CIRCLE(208,40),33,5
6820 CIRCLE(208,40),30,5
6840 PAINT(239,40),5
6860 LINE(187,28)-(229,52),7,B
6880 LINE(185,40)-(231,40),7
6900 LINE(208,25)-(208,54),7
```

```
6920 LINE(196,33)-(220,47),4,BF
6940 LINE(183,95)-(232,104),4,B
6960 LINE(184,96)-(231,103),3,BF
6980 LINE(183,119)-(232,128),4,B
7000 LINE(192,125)-(192,127),7
7020 LINE(200,123)-(200,127),7
7040 LINE(208,125)-(208,127),7
7060 LINE(216,123)-(216,127),7
7080 LINE(224,125)-(224,127),7
7100 LINE(183,143)-(192,152),4,B
7120 LINE(184,144)-(191,151),3,BF
7140 LINE(183,159)-(192,168),4,B
7160 LINE(184,160)-(191,167),3,BF
7180 LINE(183,179)-(232,190),15,B
7200 PSET(184,87),1:PRINT#1,"TREIB."
7220 PSET(200,145),1:PRINT#1,"M.FLUX"
7240 PSET(200,161),1:PRINT#1,"RADAR"
7260 RETURN
7280 '
7300 'Spritemuster definieren
7320 FOR I=0 TO N
7340  SP=I:GOSUB 7400
7360 NEXT I
7380 RETURN
7400 S$=""
7420 FOR J=1 TO 32
7440  READ D$
7460  S$=S$+CHR$(VAL("&H"+D$))
7480 NEXT J
7500 SPRITE$(SP)=S$
7520 RETURN
7540 '
7560 'Verzoegerungsschleife
7580 FOR J=1 TO N:NEXT J:RETURN
7600 '
7620 'PSG Kanal initialisieren
7640 SOUND 8,8:SOUND 9,&B10000:SOUND 11,0:SOUND 12,50
7660 SOUND 0,100:SOUND 1,0:SOUND 3,0
7680 RETURN
7700 '
7720 'PSG Kanaele schliessen
7740 SOUND 8,0:SOUND 9,0:SOUND 10,0
7760 RETURN
7780 '
7800 'data
7820 '
7840 'Spritemuster
7860 'Raumschiffrumpf
7880 DATA 02,02,07,07,0F,0F,2F,3F
7900 DATA 3F,3F,3E,07,03,00,00,00
7920 DATA 40,40,E0,E0,F0,F0,F4,EC
```

```
7940 DATA DC,DC,3C,E0,C0,00,00,00
7960 'Positionslichter des Schiffs
7980 DATA 02,02,04,04,09,0A,2A,2A
8000 DATA 2A,20,20,00,00,00,00,00
8020 DATA 40,00,00,00,80,00,00,00
8040 DATA 00,00,00,00,00,00,00,00
8060 'Auftankstation
8080 DATA 3F,7F,FF,FF,FF,FC,F8,F8
8100 DATA F8,F8,DC,DF,DE,DF,47,07
8120 DATA FC,FE,FF,FF,FF,3F,16,16
8140 DATA 16,16,37,EF,1F,FF,FE,FC
8160 'Schiff am Radarschirm
8180 DATA 00,00,00,00,02,02,07,0C
8200 DATA 78,1C,0F,00,00,00,00,00
8220 DATA 00,00,00,00,40,40,E0,30
8240 DATA 1E,38,F0,00,00,00,00,00
8260 'Pfeil nach unten
8280 DATA 00,00,00,00,00,00,00,00
8300 DATA 38,38,38,38,FE,7C,38,10
8320 DATA 00,00,00,00,00,00,00,00
8340 DATA 00,00,00,00,00,00,00,00
8360 'Peil nach rechts
8380 DATA 00,00,00,00,00,00,00,00
8400 DATA 08,0C,FE,FF,FE,0C,08,00
8420 DATA 00,00,00,00,00,00,00,00
8440 DATA 00,00,00,00,00,00,00,00
8460 'Pfeil nach oben
8480 DATA 00,00,00,00,00,00,00,00
8500 DATA 10,38,7C,FE,38,38,38,38
8520 DATA 00,00,00,00,00,00,00,00
8540 DATA 00,00,00,00,00,00,00,00
8560 'pfeil nach links
8580 DATA 00,00,00,00,00,00,00,00
8600 DATA 10,30,7F,FF,7F,30,10,00
8620 DATA 00,00,00,00,00,00,00,00
8640 DATA 00,00,00,00,00,00,00,00
8660 'Eisblock
8680 DATA 00,00,01,07,1F,3F,3B,3D
8700 DATA 3F,2F,0F,1D,0F,0D,00,00
8720 DATA 00,00,C0,F8,FC,FC,B8,DC
8740 DATA FC,7C,F8,F8,F0,F0,E0,00
8760 'Explosion
8780 DATA 00,03,00,08,03,19,03,28
8800 DATA 07,03,26,2C,00,09,00,00
8820 DATA 00,04,60,00,2C,80,00,26
8840 DATA 80,6C,C0,10,48,00,00,00
8860 '
8880 'Pos. und Groesse der Gebilde im Demoabschnitt
8900 DATA 40,32,15,  40,64,15
8920 DATA 40,144,23, 184,16,23
8940 DATA 208,80,31
```

12.6 Zusammenfassung

Nun haben Sie den Aufbau eines Video-Spiels studiert. Damit Sie feststellen, ob Sie alles verstanden haben, sollten Sie eine Anleitung für dieses Spiel schreiben. Haben Sie die Funktionen aller Module begriffen, werden Sie damit keine Probleme haben. In diesem Buch finden Sie keine Programmbeschreibung, weil wir Sie zwingen wollen, die Kommentare im Listing zu lesen.

Die Prozeduren für den Aufbau des Video-Spiels hängen davon ab, welche Programmiersprache man verwendet. Bis jetzt haben wir mit MSX-BASIC gearbeitet. Da MSX-BASIC ein Interpreter ist, ist er wesentlich langsamer als ein Programm, das im Assembler geschrieben wurde. In Kapitel 13 geben wir eine kleine Einführung, wie man Video-Spiele mit Assembler-Unterprogrammen verbessern kann.

13

Ein weiteres Spiel: Autorennen



13 Ein weiteres Spiel: Autorennen

In Kapitel 12 haben wir gelernt, daß zum Programmieren von Spielen zwei wichtige Schritte gehören: der Entwurf und das Programmschreiben. Wir haben eine "lustige" Programmiersprache, die wir Pseudocode genannt haben, benutzt und Flußdiagramme entwickelt, die uns beim Entwerfen des Programms "PLANET" geholfen haben. Dieser Prozeß beginnt mit der Beschreibung für die einzelnen Aktionsschritte. Schließlich haben wir die Pseudocode-Module in tatsächliche MSX-BASIC-Anweisungen umgesetzt. Jedes Modul kann einzeln getestet und von Fehlern befreit werden, bevor es mit den anderen zu einem Programm zusammengefaßt wird. Wir werden auch diesmal diese Prozedur anwenden und ein weiteres Spiel entwickeln.

13.1 Das Spiel

Wir entwerfen ein Autorennspiel. Unser Fahrer kann die Länge des Kurses auswählen und soll die Entfernung in der kürzestmöglichen Zeit zurücklegen. Es werden noch weitere Autos auf der Strecke vorhanden sein und unser Fahrer muß darauf achten, daß er mit keinem zusammenstößt.

13.2 Die Bildschirmanzeige

Auf der linken Seite des Bildschirms wollen wir den Kurs aus der Vogelperspektive zeigen. Auf der rechten Seite soll die Punktzahl zu sehen sein. Betrachten Sie dazu Bild 13.2.1.

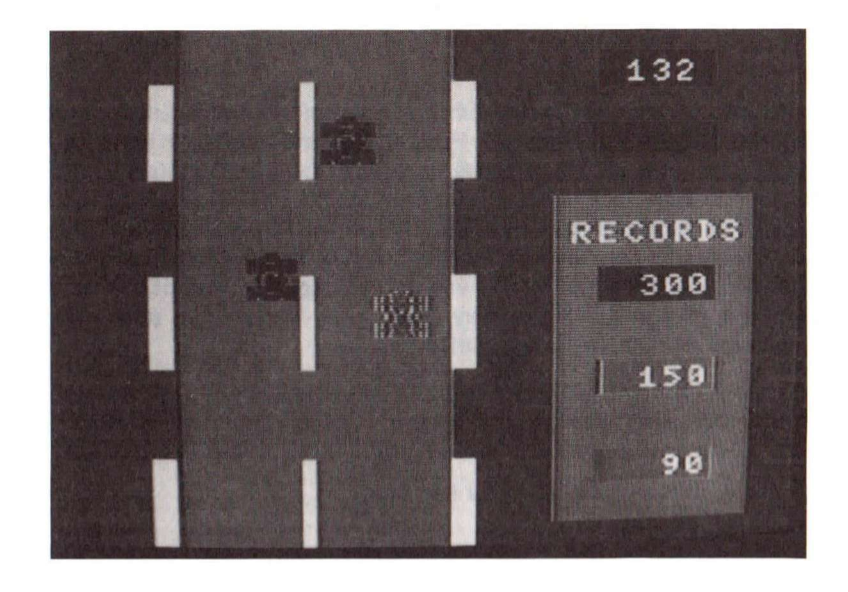


Bild 13.2.1 Autorennen

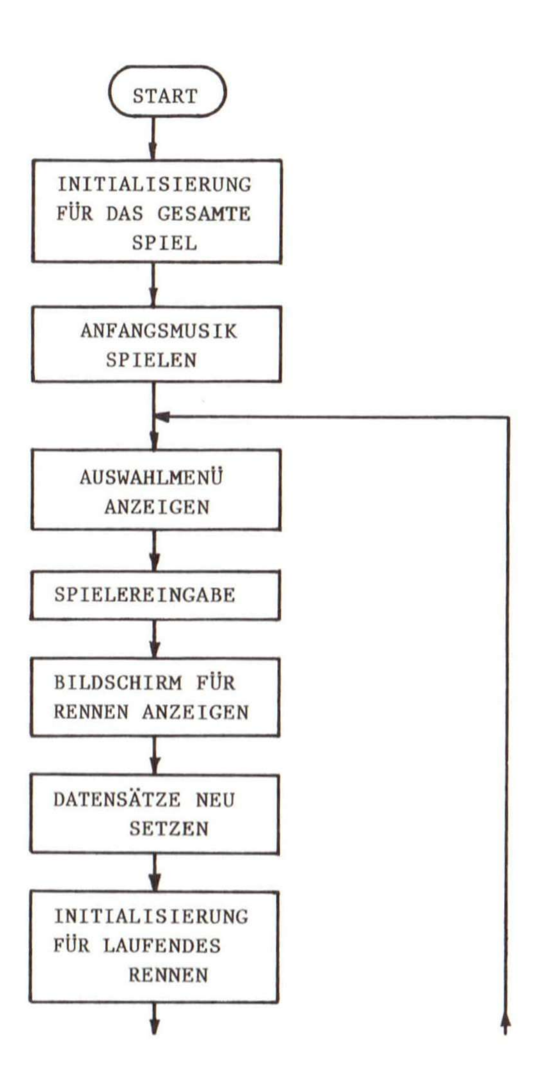
13.3 Wie man das Spiel steuert

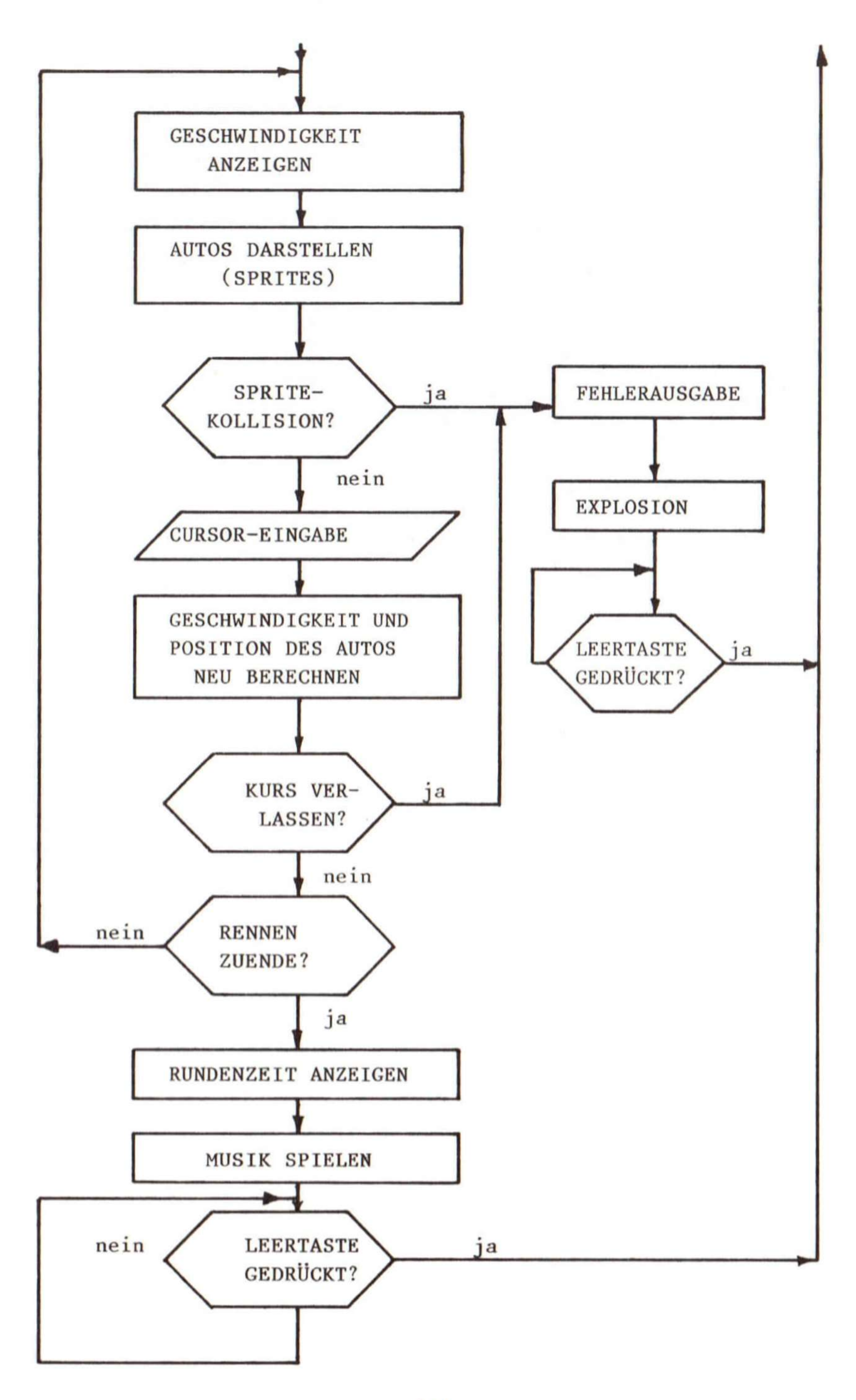
Wir verwenden die Pfeil-Tasten, um das Auto zu steuern:

^ Die Geschwindigkeit des Autos steigt an.
<- Das Auto fährt nach links.
-> Das Auto fährt nach rechts.
v Die Geschwindigkeit des Autos vermindert sich.

13.4 Einzelheiten der Programmierung

Im folgenden Flußdiagramm finden Sie die Programm-Module für dieses Spiel zusammengestellt:




GESCHWINDIGKEIT ANZEIGEN
AUTOS DARSTELLEN (SPRITES)
SPRITE-KOLLISION?
ja
FEHLERAUSGABE
nein
EXPLOSION
CURSOR-EINGABE
GESCHWINDIGKEIT UND POSITION DES AUTOS NEU BERECHNEN
LEERTASTE GEDRÜCKT?
ja
KURS VER-LASSEN?
ja
nein
nein
RENNEN ZUENDE?
ja
RUNDENZEIT ANZEIGEN
MUSIK SPIELEN
nein
LEERTASTE GEDRÜCKT?
ja

Wie in Kapitel 12 übersetzen wir dieses Flußdiagramm in eine Pseudocode-Tabelle:

Zeilennummer	Funktion
1000	Programmbeginn
1160	Definition der Sprite-Muster
1250	Auswahl der Bildschirmanzeige
1450	Beginn des Autorennens
1870	Hauptschleife für das Autorennen
2250	Abfrage auf Beendigung des Rennens
2500	Ende des Rennens
2604	Auto 1 Positions-Routine
2650	Auto 2 Positions-Routine
2700	Sprite-Muster-Routine

Auch hier zeigt die Zeilennummer wieder den Beginn eines Moduls an, das die entsprechenden Funktionen enthält. Wir verwenden folgende Variablen in unserem Programm:

Variablen-Name	Beschreibung
A$	Sprite-Muster für das Auto
B	Eingabewert des Cursors
B$	Sprite-Muster für die Explosion
C$	Zwischenzeichenkette
D	gefahrene Entfernung
DS( )	Länge des Kurses
F	waagrechte Position von Auto 1
G	senkrechte Position von Auto 1
H	Geschwindigkeit von Auto 1
P	waagrechte Position von Auto 2
Q	senkrechte Position von Auto 2
R	Geschwindigkeit von Auto 2
RC	laufende Runde
RC( )	vergangene Runde
RF	gewählte Länge des vorhergehenden Spielers
RG	gewählte Länge des jetzigen Spielers
S	Zwischenzahlen-Variable
S$	Zwischenzeichenketten-Variable
V	Geschwindigkeit des Auto 0 (Spielerauto)
X	waagrechte Position von Auto 0
Y	senkrechte Position von Auto 0
Z	Variable, die den Motorlärm erzeugt

An dieser Stelle haben wir die neuen indizierten Variablen vom Typ DS( ) und RC( ) eingeführt. Wir nennen sie Matrizen. Eine Matrix ist eine geordnete Liste von Datenelementen. Es gibt eindimensionale Listen wie:

```
A(I)
```

oder eine Tabelle von Datenelementen, die aus Reihen und Spalten bestehen wie:

```
B(I,J)
```

I und J nennt man Indizes. Bevor Sie mit einer Matrix arbeiten können, müssen Sie sie mit einer DIM-Anweisung definieren. Die DIM-Anweisung wird verwendet, um die maximale Zahl der Elemente in einer Matrix anzugeben. Das allgemeine Format einer DIM-Anweisung lautet:

```
DIM <Name>(<Ganzzahlausdruck 1>[,<Ganzzahlausdruck 2>...])
```

<Name> ist ein MSX-BASIC-Variablenname. Der Wert von <Ganzzahlausdruck> legt die maximale Zahl der Elemente für die Dimension X fest. Die maximale Anzahl der Elemente pro Dimension ist 32766.

Bei der Matrix-Definition gibt es eine Ausnahme. Erhält MSX-BASIC eine nicht definierte Variable, nimmt es eine Matrix mit einem maximalen Indexwert von 10 an. Deshalb können Matrizen auch ohne DIM-Anweisung verwendet werden.

Der minimale Indexwert für ein Matrix-Element ist 0. Die Matrix-Zuweisung:

```
DIM A(10)
```

definiert eine Matrix, die 11 Elemente besitzt:

```
A(0), A(1), A(2), ..., A(10)
```

Beachten Sie, daß jedes Element einer Matrix durch den Index einen Matrix-Namen erhält. Die Zuordnung eines Elements mit einem negativen Index löst eine "Illegal Function Call"-Fehlermeldung aus. Wird ein Matrizenelement angesprochen, dessen Index den in der DIM-Anweisung gesetzten maximalen Wert überschreitet, wird eine "Subscript out of range"-Fehlermeldung ausgegeben. Wir werden mehrdimensionale Matrizen nicht besprechen, weil dies den Rahmen dieses Buches sprengen würde.

Ohne auf die Einzelheiten näher einzugehen, zeigen wir das vollständige Programm mit den Kommentar-Anweisungen auf den nächsten Seiten. Gehen Sie jedes Modul genau durch, damit Sie dieses Programm verstehen.

```
1000 '**********
1020 '*  AUTO  *
1040 '**********
1050 '
1060 'Initialisierung
1070 DIM CL(3),DS(3),RC(3)
1080 DEFINT A-Z
1090 CLEAR 100,&HF000
1100 I=RND(-TIME)
1110 CL(1)=6:CL(2)=3:CL(3)=4
1120 DS(1)=30000:DS(2)=15000:DS(3)=9000
1130 RC(1)=300:RC(2)=150:RC(3)=90
1140 PLAY"V15T48L32O5CO4GO5ECO4GO5EO4GO5CCCC","V12T48L32O5ECO4GO5ECO4GO5CEEEE"
1150 SCREEN 2,2:OPEN"GRP:"AS#1
1160 'Spritemuster definieren
1170 GOSUB 2700:A$=S$
1180 GOSUB 2700:B$=S$
1240 '
1250 'Seite anwaehlen
1260 COLOR 15,12:SCREEN 2,2:CLS
1270 LINE(64,16)-(191,31),13,BF
1280 PSET(96,20),13:PRINT#1,"AUTORENNEN"
1290 LINE(80,80)-(97,91),6,BF
1300 LINE(80,96)-(97,107),3,BF
1310 LINE(80,112)-(97,123),4,BF
1320 PSET(68,58),12:PRINT#1,"ENTFERNUNG WAEHLEN"
1330 PSET(87,83),6:PRINT#1,"1   1000 km"
1340 PSET(87,99),3:PRINT#1,"2    500 km"
1350 PSET(87,115),4:PRINT#1,"3    300 km"
1360 RF=RG
1370 'Wahl des Spielers annehmen
1380 S$=INKEY$:IF S$<>"" THEN 1380
1390 S$=INKEY$:S=VAL(S$)
1400 ON S GOTO 1420,1420,1420
1410 GOTO 1390
1420 SOUND 7,&H38:PLAY"O4C"
1430 RG=S
1440 '
1450 'Beginn des Autorennens
1460 COLOR 15,4:SCREEN 2,2:CLS:L=5:SPRITE ON
1470 ON SPRITE GOSUB 2250
1480 SPRITE$(0)=A$
1490 SPRITE$(1)=B$
1500 LINE(0,0)-(56,191),12,BF
1510 LINE(144,0)-(255,191),12,BF
1520 LINE(176,70)-(240,183),13,BF
1530 LINE(190,46)-(226,57),CL(RG),BF
1540 LINE(190,94)-(226,105),6,BF
1550 LINE(190,126)-(226,137),3,BF
1560 LINE(190,158)-(226,169),4,BF
```

```
1570 LINE(190,22)-(226,33),1,BF
1580 LINE(190,22)-(226,33),1,BF
1590 LINE(190,46)-(226,57),CL(RG),BF
1600 PSET(182,79),13:PRINT#1,"RUND.ZT"
1610 PSET(199,25),1:PRINT#1,USING"###";0
1620 'update records
1630 IF RC<RC(RF) THEN RC(RF)=RC
1640 PSET(195,97),6:PRINT#1,USING"####";RC(1)
1650 PSET(195,129),3:PRINT#1,USING"####";RC(2)
1660 PSET(195,161),4:PRINT#1,USING"####";RC(3)
1770 'Init. Geschwindigkeit und Position
1780 X=88:Y=100:V=80
1790 F=104:G=150:H=84
1800 P=80:Q=50:R=96
1810 'Init. Maschinengeraeusch
1820 SOUND 7,&H38:SOUND 6,15
1830 SOUND 1,1:SOUND 3,2:SOUND 5,5
1840 SOUND 8,15:SOUND 9,15:SOUND 10,15
1850 RC=0:D=0
1860 '
1870 'Hauptschleife fuer Rennen
1880     RC=RC+1:D=D+V
1940     S=INT(1200/(V+1)):IF S>127 THEN S=127
1950     Z=S*2
1960     SOUND 0,Z:SOUND 4,Z:SOUND 2,Z
1970     PUT SPRITE 1,(X,Y),8,0
1980     PUT SPRITE 2,(F,G),1,0
1990     PUT SPRITE 3,(P,Q),1,0
2000     LINE(199,25)-(221,33),1,BF
2010     PSET(199,25),1:PRINT#1,USING"###";V
2020     B=STICK(0)
2030     ON B GOTO 2050,2060,2070,2080,2090,2100,2110,2120
2040     GOTO 2130
2050     V=V+4:GOTO 2130
2060     X=X+8:V=V+4:GOTO 2130
2070     X=X+8:GOTO 2130
2080     X=X+8:V=V-9:GOTO 2130
2090     V=V-9:GOTO 2130
2100     X=X-8:V=V-9:GOTO 2130
2110     X=X-8:GOTO 2130
2120     X=X-8:V=V+4:GOTO 2130
2130     IF V>200 THEN V=200
2140     G=G-(H-V)/L:Q=Q-(R-V)/L
2160     IF G>190 THEN G=5:GOSUB 2600
2170     IF Q>190 THEN Q=5:GOSUB 2650
2180     IF G<=0 THEN G=180
2190     IF Q<=0 THEN Q=180
2200     IF X<56 THEN GOSUB 2250
2210     IF X>128 THEN GOSUB 2250
2220     IF D>DS(RG) THEN 2500
```

```
2230    IF V>20 THEN 1870 ELSE V=20:GOTO 1870
2240 '
2250 'Unfall
2260 SPRITE OFF:LINE(192,22)-(226,33),1,BF
2270 PSET(195,49),15:PRINT#1,"BUMM"
2280 PSET(199,25),1:PRINT#1,USING"###";0
2290 PUT SPRITE 0,(X,Y),10,1
2300 'explosion noise
2310 SOUND 0,0:SOUND 1,5:SOUND 5,15
2320 SOUND 3,13:SOUND 4,255:SOUND 5,15
2330 SOUND 6,30:SOUND 7,0
2340 SOUND 8,16:SOUND 9,16:SOUND 10,16
2350 SOUND 11,0:SOUND 12,5:SOUND 13,0
2360 FOR I=1 TO 30:NEXT I
2370 SOUND 12,56:SOUND 13,0
2380 'Explosion anzeigen
2390 FOR I=1 TO 10
2400 PUT SPRITE 1,(X,Y),8,0:PUT SPRITE 0,(X,Y),10,1
2410 PUT SPRITE 0,(X,209):PUT SPRITE 1,(X,209)
2420 NEXT I
2430 RC=900
2440 LINE(182,8)-(233,36),10,BF
2450 PSET(188,11),10:PRINT#1,"LEER"
2460 PSET(188,19),10:PRINT#1,"TASTE"
2470 PSET(188,27),10:PRINT#1,"DRUECK"
2480 IF STRIG(0)=0 THEN 2480 ELSE SOUND 7,56:PLAY"L3205G":RETURN 1250
2490 '
2500 'Rennen beendet
2510 PSET(199,25),1:PRINT#1,USING"###";0
2520 PSET(199,49),CL(RG):PRINT#1,USING"###";RC
2530 LINE(182,8)-(233,36),10,BF
2540 PSET(188,11),10:PRINT#1,"LEER"
2550 PSET(188,19),10:PRINT#1,"TASTE"
2560 PSET(188,27),10:PRINT#1,"DRUECK"
2570 SOUND 7,56:PLAY"V14T48L3204GAB05C04G05L8C","V11T48L3203GAB04C03G0L8C",""
2580 SPRITE OFF:IF STRIG(0)=0 THEN 2580 ELSE SOUND 7,56:PLAY"L3205G":GOTO 1250
2590 '
2600 'Autoposition 1
2610 I=INT(RND(1)*4)
2620 IF I>1 THEN F=60:H=89 ELSE F=104:H=84
2630 RETURN
2640 '
2650 'Autoposition 2
2660 I=INT(RND(1)*4)
2670 IF I>1 THEN P=124:R=79:ELSE P=80:R=96
2680 RETURN
2690 '
2700 'Spriteroutine
2710 S$=""
2720 FOR I=1 TO 32
```

```
2730 READ C$:S$=S$+CHR$(VAL("&H"+C$))
2740 NEXT I
2750 RETURN
2760 '
2770 'Daten Autosprite
2780 DATA 03,07,EF,EC,FF,EF,EC,09
2790 DATA 1B,1B,0B,EF,EF,FC,EF,E7
2800 DATA C0,E0,F7,37,FF,F7,37,90
2810 DATA D8,D8,D0,F7,F7,3F,F7,E7
2820 '
2830 'Explosionssprite
2840 DATA 01,03,07,07,0F,3F,7F,FF
2850 DATA FF,7F,3F,0F,07,07,03,01
2860 DATA 80,C0,E0,E0,F0,FC,FE,FF
2870 DATA FF,FE,FC,F0,E0,E0,C0,80
```

Sie können dieses Programm eingeben oder, wenn Sie die Programmbeispielkassette besitzen, es mit CLOAD "AUTO1" von der Seite B laden. Lassen Sie das Programm laufen und spielen Sie das Autorennspiel. Achten Sie auf die Geschwindigkeit, mit der das Programm abläuft. Sie sehen selbst, daß es etwas zäh wirkt. Das ist so, weil MSX-BASIC ein Interpreter ist, der die Programmzeilen Zeile für Zeile in Maschinencode-Anweisungen umsetzen muß. Wir werden einige Maschinencodemodule einfügen und damit das Programm im nächsten Abschnitt verbessern.

13.5 Autorennen 2

Wenn Sie die Programmbeispielkassette erworben haben, lassen Sie das Programm "AUTO1" im Speicher stehen und laden das Modul "AUTOM" mit MERGE dazu. "AUTOM" finden Sie auf Seite B. Das kombinierte Programm befindet sich unter dem Namen "AUTO2" auf der gleichen Kassette.

Sonst speichern Sie "AUTO1" auf einer Kassette ab und geben Sie das untenstehende Programmodul ein. Legen Sie es unter dem Namen "AUTOM" auf Ihrer Kassette ab (mit SAVE "CAS:AUTOM"). Laden Sie nun mit CLOAD das Programm "AUTO1" (NEW nicht vergessen!) in den Speicher, und hängen Sie das Modul "AUTOM" mit MERGE dazu. Spielen Sie das Autorennen nocheinmal. Achten Sie auf die Unterschiede.

```
1190 'Maschinencoderoutine
1200 DEF USR0=&HF000
1210 FOR I=&HF000 TO &HF0AC+2+8
1220 READ C$:POKE I,VAL("&h"+C$)
1230 NEXT I
1670 'Linien fuer Strasse ziehen
1680 LINE(48,0)-(55,31),15,BF
1690 LINE(98,0)-(101,31),15,BF
1700 LINE(144,0)-(151,31),15,BF
1710 LINE(48,64)-(55,95),15,BF
1720 LINE(98,64)-(101,95),15,BF
1730 LINE(144,64)-(151,95),15,BF
1740 LINE(48,128)-(55,159),15,BF
1750 LINE(98,128)-(101,159),15,BF
1760 LINE(144,128)-(151,159),15,BF
1890    FS=3
1900    IF V>100 THEN FS=2
1910    IF V>140 THEN FS=1
1920    POKE &HF0AD,FS
1930    I=USR0(0)
2000    I=USR0(0)
2150    I=USR0(0)
2890 'Daten fuer Maschinencoderoutine data
2900 DATA 21,C0,03,CD,53,00,3A,07
2910 DATA 00,4F,3E,00,06,20,ED,79
2920 DATA 00,10,FB,21,AD,F0,35,C0
2930 DATA 3A,AE,F0,77,21,06,18,DD
2940 DATA 21,AF,F0,CD,73,F0,21,06
2950 DATA 19,DD,21,AF,F0,CD,73,F0
2960 DATA 21,06,1A,DD,21,AF,F0,CD
2970 DATA 73,F0,21,0C,18,DD,21,AF
2980 DATA F0,CD,73,F0,21,0C,19,DD
2990 DATA 21,AF,F0,CD,73,F0,21,0C
3000 DATA 1A,DD,21,AF,F0,CD,73,F0
3010 DATA 21,12,18,DD,21,AF,F0,CD
3020 DATA 73,F0,21,12,19,DD,21,AF
3030 DATA F0,CD,73,F0,21,12,1A,DD
3040 DATA 21,AF,F0,11,20,00,06,08
3050 DATA CD,50,00,3A,06,00,4F,ED
3060 DATA 78,DD,77,00,19,DD,23,10
3070 DATA EF,F5,A7,ED,52,DD,2B,DD
3080 DATA 2B,06,07,CD,53,00,3A,07
3090 DATA 00,4F,DD,7E,00,ED,79,A7
3100 DATA ED,52,DD,2B,10,ED,CD,53
3110 DATA 00,F1,ED,79,C9,01,01,00
3120 DATA 00,00,00,00,00,00,00
```

Die Verbesserungen haben wir durch die Maschinencode-Module erreicht. Diese "Codes" erhöhen die Ablaufgeschwindigkeit, weil die CPU nun nicht mehr unter Zuhilfenahme von MSX-BASIC jede Anweisung erst in die Maschinensprache umsetzen muß. Damit wir diese Module schreiben können, brauchen wir im allgemeinen eine Assembler-Sprache, die sehr stark Mikroprozessor-orientiert ist. Diese Sprache besteht aus sogenannten mnemonischen Anweisungen. Mnemonics sind Abkürzungen für englische Wörter. Ein Beispiel:

```
LD    für LOAD = laden

JP    für JUMP = springen
```

Assemblerprogramme brauchen ein besonderes Programm, das man Assembler nennt. Mit ihm wird die Sprache in für die CPU verständlichen Maschinencode umgesetzt. Ein Disassembler übersetzt die Maschinencode-Programme zurück in die Assembler-Sprache. Wir werden in diesem Buch die Assembler-Sprache nicht besprechen. Wir haben diese Maschinencode-Module nur eingeführt, um Ihnen zu zeigen, daß trotz des bereits hervorragenden MSX-BASIC immer noch Verbesserungen möglich sind.

13.6 BASIC-Wortschatz

In diesem Kapitel haben wir folgende MSX-BASIC-Wörter kennengelernt:

DIM 13.4

14

Ein Spiel für jede Jahreszeit: Bilder entwerfen



14 Ein Spiel für jede Jahreszeit: Bilder entwerfen

In diesem Kapitel weichen wir etwas von unserer bisherigen Philosophie ab, weil wir keinen Versuch machen, den Entwurf und die Codierung des Programms "MALPRG" zu zeigen. Das vollständige Programm, das am Ende dieses Kapitels aufgelistet wird, ist für diejenigen vorgesehen, die der Assembler-Sprache mächtig sind.

Wir haben "MALPRG" aus zwei Gründen in dieses Buch mit aufgenommen:

1. damit Sie Spaß haben,
2. um Ihnen zu zeigen, wie Assemblerprogramme die Ablaufgeschwindigkeit erhöhen.

Für dieses Programm benötigen Sie 32 KByte RAM. Besitzen Sie die Programmbeispielkassette, kann es mit CLOAD von Seite B geladen werden. Mit "MALPRG" können Sie Farbskizzen mit Ihrem MSX-Computer erstellen. Lassen Sie Ihr künstlerisches Talent zum Vorschein kommen und versuchen Sie sich in Computerkunst.

14.1 Wie man das Programm benutzt

Wenn "MALPRG" geladen ist, und Sie es mit RUN gestartet haben, sehen Sie auf Ihrem Schirm folgendes Bild:

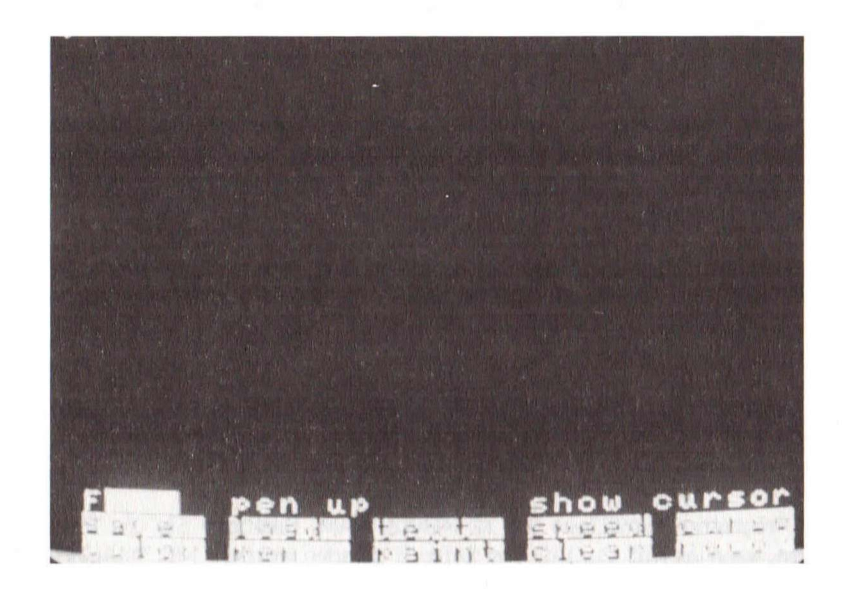


Bild 14.1.1 MALPRG

Der obere Teil des Bildschirm ist Ihr Arbeitsbereich, in dem Sie Ihre Bilder erstellen können.

Die dritte Zeile von unten ist die Statuszeile. Sie zeigt an, welche Funktionen aktiv sind. Haben Sie eine deutsche Programmversion, stimmt der angezeigte Text auf dem Bild mit dem des Programms nicht überein. Um Unklarheiten zu vermeiden, finden Sie anschließend die zugeordneten deutschen Bezeichnungen:

pen up = Stift oben	show cursor = Cursor da
save = Speic	load = Laden
text = Text	speed = Gesch
curso = Curso	
color = Farbe	pen = Stift
paint = Malen	clear = Losch
recov = Rueck	

Die Leertaste ist ein Schalter, mit dem Sie den Cursor (einen Stern) anzeigen oder verschwinden lassen können. Er gibt die Position Ihres Stiftes am Bildschirm an. Drücken Sie die Leertaste und achten Sie auf die Statuszeile, die die Cursor-Funktion anzeigt.

Sie können den Cursor innerhalb des Arbeitsbereichs mit den Pfeil-Tasten bewegen. Ist die Position Ihres "Bleistifts" oben, wird nichts gezeichnet. Ist der Stift unten, wird eine Linie gezeichnet. Mit der Funktionstaste F2 können Sie Ihren Bleistift nach unten oder nach oben setzen. Die Statuszeile zeigt diese Funktion wieder mit Stift oben oder Stift unten an.

Die untersten beiden Zeilen sind die Bezeichnungen für die Funktionstasten F1 - F10. Wir werden die Funktionen in der folgenden Tabelle erklären:

Taste	**Bezeichng**	**Funktion**
F1	Farbe	zeigt die 16 Farben an, mit denen Sie zeichnen oder malen können. Sie müssen nur die Taste drücken, die der gewünschten Farbe entspricht und sofort wird die Farbe für das Zeichnen umgeschaltet.
F2	Stift	Ihr Stift wird nach oben oder unten gesetzt. Ist der Bleistift unten, wird eine Linie gezeichnet, wenn der Cursor sich bewegt. Ist der Bleistift oben, können Sie ihn von einer Position zur anderen bringen, ohne eine Linie zu zeichnen.
F3	Malen	Malen füllt ein Bild mit der gewünschten Farbe aus. Die Figur muß vollständig (in der Füllfarbe) umrahmt sein, sonst wird über den Bildrand hinaus gemalt. Um Malen zu verwenden, müssen Sie: 1. Die gewünschte Farbe anwählen, 2. den Rahmen zeichnen, 3. den Cursor mit Stift oben in die Mitte des Bildes stellen, das ausgemalt werden soll. Dabei darf der Bleistift keine Rahmenlinie berühren. 4. F3 drücken, um Malen aufzurufen.
F4	Losch	löscht den Arbeitsbereich für eine weitere Skizze.

Taste	Bezeichng	Funktion
F5	Rueck	Rueck ist die Abkürzung für rückgängig machen. Haben Sie Ihren Entwurf aus Versehen gelöscht, wird mit Rueck der Fehler wieder zurückgenommen und Ihre Arbeit erneut gezeigt. Wird beim Malen-Befehl mehr als die gewünschte Figur ausgefüllt, können Sie mit Rueck den Vorgang stoppen und das Bild wird wieder im alten Zustand gezeigt.
F6	Speic	Der Entwurf im Speicher wird auf die Kassette abgespeichert. Der Computer fragt Sie nach einem Namen für Ihren Entwurf, der sechs Zeichen groß sein darf. Schalten Sie dann Ihren Recorder ein und drücken Sie RETURN. Wird die Skizze wieder angezeigt, haben Sie eine Kopie auf der Kassette erstellt.
F7	Laden	Ein vorher abgespeicherter Entwurf wird wieder von der Kassette in den Speicher zurückgeladen. Der Computer fragt Sie nach dem Namen der Skizze. Legen Sie eine Kassette ein, schalten Sie in den Aufnahme-Modus und drücken Sie RETURN. Wenn der Vorgang beendet ist, wird der Entwurf gezeigt. Ein vorher im Speicher stehender Entwurf wird überladen und ist damit verloren.
F8	Text	aktiviert den Textschreib-Modus, der es möglich macht, daß Buchstaben in den Entwurf geschrieben werden können.
F9	Gesch	aktiviert den Modus, mit dem Sie die Geschwindigkeit für die Cursorbewegung anwählen können. Sie sehen eine Skala von 1 bis 9. Drücken Sie die gewünschte Zahl. (1 steht für die niedrigste, 9 für die höchste Geschwindigkeit.)
F10	Curso	Eine Abkürzung für Cursor. Damit wird die Cursoranzeige umgeschaltet. Bewegt sich der Cursor in ein Farbgebiet, in dem er unsichtbar ist, wird eine andere Farbe angewählt, damit er wieder sichtbar wird.

Zusammen mit den Pfeil-Tasten und den obigen Anweisungen können Sie die tollsten Gemälde entwerfen und sie für den späteren Gebrauch abspeichern.

Im nächsten Abschnitt finden Sie das komplette Programmlisting von "MALPRG".

14.2 Das Programmlisting

```
1000 '*****************
1002 '*    MALPRG      *
1004 '*****************
1005 '
1008 CLEAR 200,&HBFFF
1010 ON STOP GOSUB 30000:STOP ON
1020 ON ERROR GOTO 25000
1179 '
1180 'initialisierung
1190 FOR I=1 TO 10:KEY I,"":NEXT I
1200 FC=15:BC=1:BD=7:COLOR FC,BC,BD:SCREEN 2,0
1220 OPEN"GRP:"AS#1
1240 B$="":FOR I=1 TO 8:READ A$:B$=B$+CHR$(VAL("&H"+A$)):NEXT I:SPRITE$(0)=B$
1250 B$="":FOR I=1 TO 8:READ A$:B$=B$+CHR$(VAL("&H"+A$)):NEXT I:SPRITE$(1)=B$
1252 'Daten fuer Cursor
1254 DATA 10,10,28,C6,28,10,10,00
1255 DATA 00,00,00,00,00,FC,FC,00
1260 DW=0:TC=15:TC$="F"
1280 PE=0:JY=-1
1285 LS=4:GOSUB 9118
1290 XN=12:YN=8
1300 C1=15:C2=15:CC=C2:IK$="F"
1320 GOSUB 1980:GOSUB 1820
1340 SS=0:GOSUB 2060
1360 GOSUB 9590:STRIG(0) ON
1380 ON KEY GOSUB 5000,5500,6000,6500,7000,7500,8000,8500,9000,9500
1400 ON STRIG GOSUB 9530
1419 '
1420 '********  Hauptprogramm  ********
1440 IF JY THEN GOSUB 1580:GOTO 1440 ELSE 1440
1480 PUT SPRITE 0,(XN-3,YN-4),CC,0
1500 IF PE AND DW THEN LINE(XO,YO)-(XN,YN),C1
1520 XO=XN:YO=YN
1540 PE=-1
1560 RETURN
1579 '
1580 'Joystickroutine
1600 ST=STICK(0):IF ST=0 THEN PUT SPRITE 0,(XN-3,YN-4),CC,0
1620 XN=XN+JX(ST):IF XN>255 THEN XN=255 ELSE IF XN<0 THEN XN=0
1640 YN=YN+JY(ST):IF YN>167 THEN YN=167 ELSE IF YN<0 THEN YN=0
1660 GOSUB 1480
1680 RETURN
1799 '
1800 '*********  Unterprogramme  ********
1820 'Init.Funktionstasten
1830 RESTORE 1860
1840 FOR I=1 TO 10:READ KY$(I):NEXT I
1860 DATA Speic,Laden,Text,Gschw,Curso,Farbe,Stift,Malen,Losch,Rueck
1879 '
1880 'Funktionstasten anzeigen
```

```
1900 GOSUB 9640:COLOR BC,FC:I=1
1920 FOR X=16 TO 208 STEP 48:LINE(X,177)-(X+39,183),FC,BF
1930 PRESET(X+1,177):PRINT#1,KY$(I):I=I+1:NEXT X
1940 FOR X=16 TO 208 STEP 48:LINE(X,185)-(X+39,191),FC,BF
1950 PRESET(X+1,185):PRINT#1,KY$(I):I=I+1:NEXT X
1960 COLOR FC,BC:GOSUB 9590:RETURN
1979 '
1980 'Init.momentanes Farbfeld
2000 LINE(16,168)-(23,175),BC,BF:PRESET(17,169):PRINT#1,IK$
2020 LINE(24,168)-(47,175),C1,BF:LINE(24,168)-(47,175),15,B
2040 RETURN
2059 '
2060 'Init.VRAM Machinecoderoutine
2070 RESTORE 2160
2080 FOR I=&HF000 TO &HF05F
2100 READ A$:POKE I,VAL("&H"+A$):NEXT I
2120 DEFUSR0=&HF000:DEFUSR1=&HF030
2140 RETURN
2160 DATA F3,3A,06,00,4F,21,00,00,CD,50,00,21,00,C0,ED,78
2180 DATA 77,23,7C,FE,D5,C2,0E,F0,21,00,20,CD,50,00,21,00
2200 DATA D5,ED,78,77,23,7C,FE,EA,C2,21,F0,FB,C9,00,00,00
2220 DATA F3,3A,07,00,4F,21,00,00,CD,53,00,21,00,C0,7E,ED
2240 DATA 79,23,7C,FE,D5,C2,3E,F0,21,00,20,CD,53,00,21,00
2260 DATA D5,7E,ED,79,23,7C,FE,EA,C2,51,F0,FB,C9,00,00,00
2319 '
2320 'Init.Farbauswahllinie
2340 CL=0:RESTORE 2500:LINE(0,184)-(255,191),BC,BF
2360 FOR I=0 TO 240 STEP 16
2380 PRESET(I+1,184):READ P$:PRINT#1,P$
2400 LINE(I+8,184)-(I+15,191),CL,BF
2420 LINE(I+8,184)-(I+15,191),15,B
2440 CL=CL+1
2460 NEXT I
2480 RETURN
2500 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
2519 '
2520 '** Funktionstasten Unterprograme **
5000 'Farbaenderroutine
5020 GOSUB 20001:STRIG(0) ON:GOSUB 2320
5040 GOSUB 9942
5120 IK$=A$:C1=VAL("&H"+IK$):GOSUB 1980
5140 LINE(0,184)-(255,191),BC,BF:GOSUB 1880
5160 GOSUB 9590:RETURN
5180 LINE(0,184)-(255,191),BC,BF:GOSUB 1880
5495 '
5500 'Zeichnen-ein/aus-Routine
5510 GOSUB 9640:DW=NOT DW
5520 LINE(60,168)-(150,175),BC,BF:PRESET(65,168)
5530 IF DW THEN PRINT#1,"Stift unten" ELSE PRINT#1,"Stift oben"
5540 GOSUB 9590:RETURN
```

```
5995 '
6000 'Malroutine
6010 GOSUB 9640:IF NOT DW AND PE THEN A=USR0(0):SS=-1:PAINT(XN,YN),C1
6020 GOSUB 9590:RETURN
6495 '
6500 'Loeschroutine
6510 GOSUB 9640:A=USR0(0):SS=-1:CLS
6518 'draw off; stick on; trigger off
6520 GOSUB 1980:DW=-1:GOSUB 5500:JY=0:GOSUB 9530:GOSUB 1880:RETURN
6995 '
7000 'Rueckgaengigmachen
7010 GOSUB 9640:IF SS THEN A=USR1(0):SS=0
7020 LINE(0,168)-(255,192),1,BF
7030 GOSUB 1980:DW=NOT DW:GOSUB 5500:JY=NOT JY:GOSUB 9530:GOSUB 1880:RETURN
7495 '
7500 'auf Kassette speichern
7510 GOSUB 20001
7520 A=USR0(0):SS=-1:GOSUB 9690
7530 GOSUB 7000
7540 GOSUB 9590:STRIG(0) ON:RETURN
7995 '
8000 'von der Kassette laden
8010 GOSUB 20001
8020 A=USR0(0):SS=-1:GOSUB 9820
8030 GOSUB 7000
8040 GOSUB 9590:STRIG(0) ON:RETURN
8495 '
8500 'Textschreibroutine
8510 GOSUB 20001:A=USR0(0):SS=-1
8530 LINE(0,168)-(255,192),13,BF
8535 GOSUB 8800
8550 HP=XN:VP=YN:PUT SPRITE 0,(HP,VP),TC,1
8560 IF VP>160 THEN 8630
8565 COLOR TC
8570 I$=INKEY$:IF I$="" THEN 8570
8580 I=ASC(I$):IF I=&H7F THEN 8570
8585 IF I=24 THEN GOSUB 8700:GOTO 8570
8590 IF I=27 THEN COLOR 15:GOSUB 7000:STRIG(0) ON:RETURN
8595 IF I=13 THEN COLOR 15:A=USR0(0):GOSUB 7000:STRIG(0) ON:RETURN
8600 IF I<&H20 THEN 8570
8620 PSET(HP,VP),POINT(HP,VP):HP=HP+8:PRINT #1,I$
8622 IF HP>255 THEN VP=VP+8:HP=HP-256
8623 IF VP>160 THEN 8630 ELSE PUT SPRITE 0,(HP,VP),TC,1:GOTO 8570
8625 '
8630 LINE(0,168)-(255,179),9,BF
8635 COLOR 15 :PUT SPRITE 0,(HP,VP),0,1
8640 BEEP:PSET(10,170),9:PRINT#1,"Cursor aus dem SCHREIB-Bereich!"
8645 I$=INKEY$:IF I$="" THEN 8645 ELSE I=ASC(I$)
8655 IF I=27 THEN 8590
8660 IF I=13 THEN 8595 ELSE 8645
```

```
8669 '
8700 COLOR 15:LINE(0,168)-(255,183),13,BF
8705 PSET(10,170),13:PRINT#1,"momentane Farbe ist"TC$
8710 LINE(160,170)-(167,177),TC,BF
8715 LINE(160,170)-(167,177),15,B
8720 GOSUB 2340:GOSUB 9942:TC$=A$:TC=VAL("&H"+A$):GOSUB 8800
8725 COLOR TC:PUT SPRITE 0,(HP,VP),TC,1:RETURN
8799 '
8800 LINE(0,168)-(255,192),13,BF
8810 PSET(10,170),13:PRINT#1,"SCHREIBMODUS      SELECT Farbe"
8820 PSET(10,183),13:PRINT#1,"ESC f. rueckg.    CR zurueck"
8830 RETURN
8995 '
9000 'Cursorgeschwindigkeit setzen
9001 GOSUB 20001:STRIG(0) OFF
9002 LINE(0,168)-(255,192),6,BF
9003 PSET(10,170),6:PRINT#1,"Cursorgeschw. setzen (1-9)"
9004 PSET(10,183),6:PRINT#1,"im Moment"LS
9010 I$=INKEY$:IF I$="" THEN 9010
9020 I=ASC(I$)
9030 IF I<49 OR I>57 THEN 9010
9040 LS=I-48
9118 RESTORE 9191:GOTO 9150
9150 FOR I=1 TO 8:READ DX,DY:JX(I)=DX*LS:JY(I)=DY*LS:NEXT I
9160 SS=0:GOSUB 7000:STRIG(0) ON:RETURN
9191 DATA 0,-1, 1,-1, 1,0, 1,1, 0,1, -1,1, -1,0, -1,-1
9495 '
9500 'Cursorfarbaenderroutine
9510 GOSUB 9640:IF C2=15 THEN C2=1 ELSE C2=15
9520 GOSUB 9560:GOSUB 9560:GOSUB 9590:RETURN
9529 '
9530 'strig on routine
9540 LINE(160,168)-(255,175),BC,BF
9550 IF JY THEN PRESET(161,168):PRINT#1,"Cursor weg"
9555 IF NOT JY THEN PRESET(161,168):PRINT#1,"Cursor da"
9560 JY=NOT JY:CC=C2*(-JY)
9570 PUT SPRITE 0,(XN-3,YN-4),CC,0
9580 RETURN
9589 '
9590 'Key on fuer Screen 2
9600 VDP(1)=&HC0:FOR J=1 TO 10:KEY(J) ON:NEXT J:VDP(1)=&HE0:RETURN
9610 'key on for screen 0
9620 VDP(1)=&HD0:FOR J=1 TO 10:KEY(J) ON:NEXT J:VDP(1)=&HF0:RETURN
9639 '
9640 'Key off fuer Screen 2
9650 VDP(1)=&HC0:FOR J=1 TO 10:KEY(J) OFF:NEXT J:VDP(1)=&HE0:RETURN
9660 'key off for screen 0
9670 VDP(1)=&HD0:FOR J=1 TO 10:KEY(J) OFF:NEXT J:VDP(1)=&HF0:RETURN
9689 '
9690 'auf Kassette speichern
```

```
9700 ON ERROR GOTO 9950
9710 COLOR ,4:SCREEN 0:KEY OFF
9720 LOCATE 3,5:PRINT"Bild auf Kassette speichern"
9730 LOCATE 3,20:PRINT"ESC fuer rueckgaengig"
9740 LOCATE 3,9:PRINT"Dateiname? ";:GOSUB 9990
9750 IF NC<>-1 THEN 9790
9760 LOCATE 3,11:PRINT"Abspeichern..."
9770 LOCATE 3,20:PRINT"                         "
9780 BSAVE"cas:"+NM$,&HC000,&HE9FF
9790 ON ERROR GOTO 25000
9800 COLOR ,1:SCREEN 2
9810 RETURN
9819 '
9820 'von der Kassette laden
9830 ON ERROR GOTO 9950
9840 COLOR ,12:SCREEN 0:KEY OFF
9850 LOCATE 3,5:PRINT"Bild von Kassette laden"
9860 LOCATE 3,20:PRINT"ESC fuer rueckgaengig"
9870 LOCATE 3,9:PRINT"Dateiname? ";:GOSUB 9990
9880 IF NC<>-1 THEN 9920
9890 LOCATE 3,11:PRINT"Suchen und Laden..."
9900 LOCATE 3,20:PRINT"                         "
9910 BLOAD"cas:"+NM$
9920 ON ERROR GOTO 25000
9930 COLOR ,1:SCREEN 2
9940 RETURN
9941 '
9942 'Farbcoderoutine
9943 A$=INKEY$:IF A$="" THEN 9943 ELSE ID=ASC(A$)
9944 IF ID>=48 AND ID<=57 THEN RETURN
9945 IF ID>=65 AND ID<=70 THEN RETURN
9946 IF ID>=97 AND ID<=102 THEN A$=CHR$(ID-32):RETURN ELSE 9943
9949 '
9950 'Fehlerbehandlung
9960 LOCATE 8,15:PRINT "Device I/O error!"
9970 FOR J=1 TO 1000:NEXT J
9980 RESUME NEXT
9990 'Eingabestring annehmen
10000 VDP(1)=&HD0
10010 FOR J=1 TO 10:KEY(J) OFF:NEXT J
10020 VDP(1)=&HF0
10030 NM$="":NC=0:CP=POS(0)
10040 I$=INKEY$:IF I$="" THEN 10040
10050 I=ASC(I$):IF I=27 THEN RETURN
10060 IF I>27 AND I<32 OR I=127 THEN 10040
10070 IF I>31 THEN IF NC>5 THEN 10040 ELSE PRINT I$;:NM$=NM$+I$:NC=NC+1:GOTO 10040
10080 IF I=8 AND POS(0)=CP THEN 10040
10085 IF I=8 AND POS(0)<>CP THEN PRINT CHR$(127);:NC=NC-1:NM$=LEFT$(NM$,NC):GOTO 10040
10090 IF I<>13 THEN 10040 ELSE NC=-1:RETURN
```

```
20000 '
20001 'Funk.tasten sperren, strig; Tastaturpuffer loschen
20010 GOSUB 9640:STRIG(0) OFF
20020 I$=INKEY$:IF I$<>"" THEN 20020
20030 RETURN
25000 'Programmfehlerroutine
25005 COLOR 15,12:SCREEN 0
25010 PRINT "ERR ="ERR
25015 PRINT "ERL ="ERL:PRINT
25020 GOTO 30020
30000 'Textmodus ruecksetzen
30010 COLOR 15,12:SCREEN 0
30020 RESTORE 30120
30030 FOR I=1 TO 8
30040 READ I$
30050 IF RIGHT$(I$,1)="*" THEN I$=LEFT$(I$,LEN(I$)-1)+CHR$(13)
30060 KEY I,I$
30070 NEXT I
30080 KEY 9,"list."+CHR$(13)+CHR$(30)+CHR$(30)
30090 KEY 10,CHR$(12)+"run"+CHR$(13)
30100 KEY ON
30110 STOP
30120 DATA "color ","auto ","goto ","list ","run*"
30130 DATA "color  15,4,7*",cload","cont*"
```

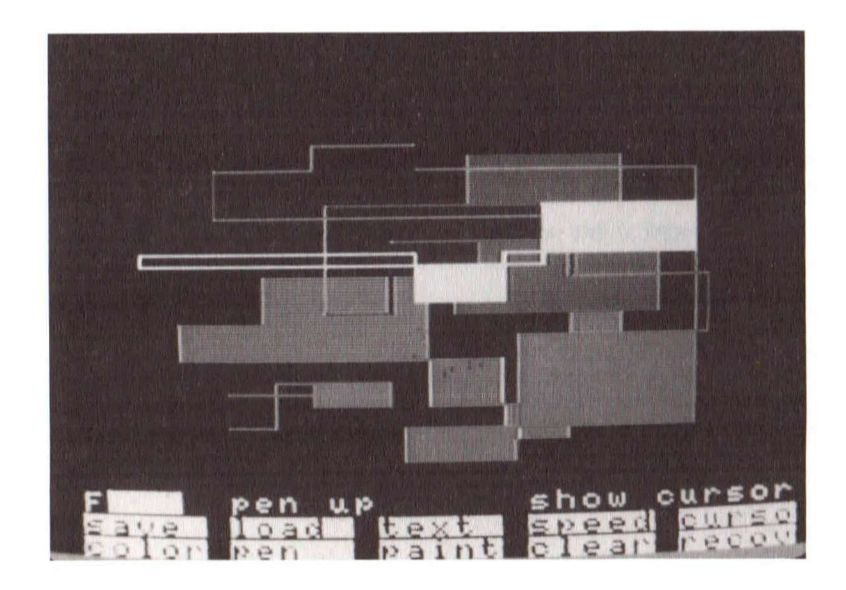


Bild 14.2.1 MALPRG-Beispiel

15

Begriffslexikon



15 Begriffslexikon

Im folgenden finden Sie eine alphabetische Liste der meisten Computer-Fachbegriffe:

Adresse	Den Hauptspeicher eines Computers kann man sich wie aufgestapelte Schachteln vorstellen. Jeder Speicherplatz (oder jede Schachtel) hat eine Zahl als Adresse.
Alphanumerisch	eine Gruppe von Zeichen, die alphabetische Buchstaben und Ziffern enthalten.
Anweisung	der kleinste vollständige Abschnitt eines Programms. Sie fängt mit einem Befehl an. Der Befehl kann Ausdrücke enthalten.
Anwenderprogramm	Programm für einen bestimmten Zweck, zum Beispiel Textverarbeitung, Tabellenkalkulation usw.
Anwenderfreundlich	ein Computersystem oder ein Programm, das für Computerneulinge leicht zu handhaben und zu verstehen ist.
Anwenderhandbuch	ein Buch, das beschreibt, wie man einen bestimmten Teil der Ausrüstung oder ein Programm handhabt.
Argument	die Variable, Zahl oder Zeichenkette, die zwischen den Klammern einer Funktion erscheint. Wie:

```
INT(N)    hat N    als Argument
LEN(W$)   hat W$   als Argument
```

Arithmetischer Ausdruck	eine Anzahl Buchstaben, Zahlen und/oder Symbole, die den Computer veranlassen, eine arithmetische Funktion durchzuführen. Beispiele:

```
2+2
2*2
A^22
2/4
2/a
A*(2/B^8)
```

Arithmetischer Operator ein Symbol, das den Computer veranlaßt, eine arithmetische Operation auszuführen. Es gibt folgende Operatoren: + (Addition), - (Subtraktion), * (Multiplikation), / (Division), ^ (Potenzierung).

ASCII steht für American Standard Code for Information Interchange (= Amerikanischer Standard-Code für Informationsaustausch). Jedem Zeichen ist eine ASCII-Zahl zugeordnet.

Assembler Programm, das ein in symbolischer Maschinensprache geschriebenes Programm in "ausführbaren" Objektkode umsetzt.

Assemblersprache eine Programmiersprache, die wesentlich schneller ist, als eine Hochsprache wie BASIC. Maschinensprachprogramme sind sehr schwierig zu schreiben. Anschließend sehen Sie zwei Zeilen eines Maschinensprachprogramms:

```
LDA 2000H
MOV C,A
```

Auflistung eine Liste aller Zeilen eines Programms.

Auflösung die Anzahl der Punkte, die man vertikal und horizontal auf einem Fernsehgerät (oder Monitor) darstellen kann. Der Ausdruck: hochauflösend gibt an, daß eine große Anzahl Punkte dargestellt werden kann.

Aufruf Die Verwendung von GOSUB ruft ein Unterprogramm auf. Das Setzen einer Funktion in eine Anweisung ruft diese Funktion auf. Unter Aufruf versteht man, daß der Computer die Befehle im Unterprogramm oder den Rechenvorgang für die Funktion aufruft und dann hinter den Aufruf zurückkehrt.

Ausdruck der Teil der Anweisung, der einen einzelnen Wert - entweder eine Zahl oder eine Zeichenkette - enthält. Beispiele:

```
7*X=1                   3*LEN(D$)=RND(1)
GT$=LEFT$(D$,2)         "APFEL"=N$
```

Ausführung Ein Programm laufen lassen oder einen einzelnen Befehl oder eine Anweisung ausführen.

Ausgabe	Daten, die vom internen Speicher des Computers zu externen Geräten übertragen werden.
Ausgabegerät	Gerät, das Rechnerinformationen anzeigt, speichert oder ausdruckt. Video-Monitore, Massenspeicher (zum Beispiel ein Diskettenlaufwerk) und Drucker sind Augabegeräte.
BASIC	Beginners All-purpose Symbolic Instruction Code. Eine höhere Programmiersprache, die von John Kemeny und Thomas Kurtz von der Dartmouth Universität in den frühen sechziger Jahren entwickelt wurde und leicht zu lernen ist.
Baud	Einheit für die Geschwindigkeit der seriellen Datenübertragung. Bei Mikrocomputern entspricht ein Baud etwa einem Bit pro Sekunde (abgekürzt: BPS).
Baudrate	Übertragungsrate für den Informationsaustausch zwischen Rechnern. 300 Baud entspricht einer Übertragungsrate von 300 Bits pro Sekunde. Nehmen wir an, daß Sie Text senden. Im Durchschnitt enthält jedes Wort sechs Zeichen und nach jedem Wort folgt ein Leerzeichen. Dann werden etwa fünf Wörter in der Sekunde oder 300 in der Minute übertragen. Terminals oder Drucker werden oft mit 4800 oder 9600 Baud betrieben.
Befehl	In BASIC bringt ein Befehl den Computer dazu, etwas zu tun; zum Beispiel: den Speicher mit dem NEW-Befehl löschen. Siehe Anweisung, Ausdruck.
Bemerkung	eine kurze Anmerkung, die ins Programm mit Hilfe der REM-Anweisung oder des Apostrophs (') gesetzt wird. Ein Beispiel: `REM Das ist eine Bemerkung`
Bereitschaftsanzeige	ein Symbol (ein Größerzeichen (>) oder die Meldung Ok), das angibt, daß der Computer auf eine Eingabe wartet.
Betriebssystem	eine Anzahl von Programmen, die eine Reihe von Geräten ansprechen und sie zu einem Rechnersystem zusammenfassen. Das Betriebssystem muß im Computer vorhanden sein, bevor Anwenderprogramme geladen oder ausgeführt werden können.

Bildschirm	der Fernsehbildschirm oder der eines Monitors, der mit dem Computer verbunden ist. Siehe Monitor.
Binärsystem	ein Zahlensystem, das nur zwei Ziffern verwendet, nämlich 0 und 1, um damit alle numerischen Werte auszudrücken. Jede Stelle wird als Potenz zur Basis 2 dargestellt.
Bit	das Grundelement des Computerspeichers. Es kann die Werte 0 oder 1 annehmen.
Boot	bedeutet, daß nach dem Einschalten des Computers ein Programm abläuft, das die notwendigen Schritte einleitet, damit der Computer mit dem Anwender "reden" kann. Eine leichte Sache für moderne Computer. Früher war das ein komplizierter Vorgang. Heute bedeutet es normalerweise das Einlesen des Betriebssystems von einer Diskette (englisch: Disk Operating System oder abgekürzt DOS).
BPS	eine Abkürzung für Bits pro Sekunde. Siehe auch Baud und Baudrate.
Bus	Über den BUS werden Informationen zwischen den Bausteinen eines Computersystems ausgetauscht.
Byte	eine binäre Einheit, die im allgemeinen aus 8 Bit besteht. Der Code eines Zeichens wird normalerweise in einem Byte abgelegt.
Cartridge	eine kleine ROM-Platine, die ein Programm enthält. Die Platine wird in den Einsteck-Schacht gesteckt. Siehe auch ROM-Modul.
Chip	ein geformtes Stück Silizium, das integrierte Schaltkreise enthält.
Code	ein System aus Symbolen und Regeln, um Daten darzustellen, zu übertragen und abzuspeichern.
Compiler	übergeordnetes Programm, das in Hochsprache geschriebene Programme (BASIC, FORTRAN, PASCAL) in maschinenorientierte Sprachen umsetzt. Die Ablaufgeschwindigkeit des Programms vergrößert sich.
Computer	eine CPU mit Speichern und Ein/Ausgabekanälen.

CP/M	(Control program for microcomputers) ein besonderes Betriebssystem, das auf vielen verschiedenen Computern läuft. Es wurde von Digital Research geschrieben.
CPU	Abkürzung für Central Processing Unit = Zentraleinheit.
CRLF	Abkürzung für carriage return (=Wagenrücklauf) gefolgt von line feed (=Zeilenvorschub). Das entspricht dem "Wagenrücklauf" einer Schreibmaschine. Siehe Wagenrücklauf, Zeilenvorschub.
CRT	Abkürzung für Cathode Ray Tube = Kathodenstrahlröhre. Fernsehgeräte oder Monitore enthalten CRTs.
Cursor	sichtbares Zeichen, das die Position am Bildschirm anzeigt. Meistens wird es als blinkendes Quadrat oder blinkender Strich dargestellt. Der Cursor zeigt an, wo das nächste Zeichen auf dem Bildschirm erscheint.
Datei	eine Anzahl von Datensätzen (records), die sich auf einer Kassette, einer Diskette oder im Speicher des Computers befinden.
Daten	alle Arten von Informationen, die ein Computer ausführen oder erzeugen kann - Zahlen, Buchstaben, Symbole, Anweisungen, ...
Datensatz	ein organisierter Datenblock (englisch: record).
Datenübertragung	siehe Baudrate
Datenverarbeitung	der Vorgang, bei dem die Daten in maschinenlesbare Form gebracht werden, so daß der Computer damit arbeiten kann.
Dezimales System	Zahlensystem, in dem jede Stelle als Potenz zur Basis 10 dargestellt wird. Die Ziffern 0 bis 9 werden verwendet.
Dienstprogramm	Programm, das dem Anwender beim Arbeiten mit einem Rechnersystem hilft.
Digitalsystem	siehe Binärsystem und Bit
Digital-Computer	ein Rechner, der den Zustand von "Spannung-ein" und "Spannung-aus" verwendet, um Daten darzustellen. Diese Schaltzustände werden in Binärzahlen umgewandelt.

Diskette	eine magnetische Scheibe in einer Schutzhülle. Disketten werden als Massenspeicher bei Personalcomputern eingesetzt.
Diskettenbetriebssystem	ein Betriebssystem, bei dem der Computer ein oder mehrere Diskettenlaufwerke verwenden kann. Siehe Betriebssystem.
Diskettenlaufwerk	ein elektromechanisches Gerät, das Daten auf einer Diskette speichert oder von ihr holt.
Dokumentation	alle verfügbaren Daten über einen bestimmten Computer, ein Programm oder ein Programmpaket, zum Beispiel: wie schaltet man den Computer ein, wie lädt man Programme, usw.
Doppelte Dichte	eine Technik, bei der man doppelt soviel Daten auf einer Diskette abspeichern kann als bei einem System mit einfacher Dichte.
DOS	Abkürzung für Disk Operating System = Diskettenbetriebssystem.
Drucker	ein Gerät, das Daten aus dem Rechner auf Papier druckt.
Dynamischer Speicher	Speicherung von Informationen als elektrische Ladung, die verloren gehen können und deshalb immer wieder neu aufgefrischt werden müssen.
Editieren	die Daten am Bildschirm oder in einem Programm ändern.
Editor	Programm zum Erstellen und Ändern von Textdateien.
Eingabe	die Datenübertragung von der Tastatur oder von einem Massenspeicher in den Speicher des Computers.
Eingabegerät	ein Gerät, mit dem man Daten in den Computer bringt. Beispiele: Tastatur, Joystick, Diskettenlaufwerk, Kassetten-Recorder.
Ein/Ausgabe	der Vorgang der Ein- oder Ausgabe von Daten zum und vom Computer.

Elektronische Nachricht	eine Nachricht wird zu einem anderen Rechner übertragen oder kommt von einem anderen Rechner. Die Computer müssen untereinander (zum Beispiel) über eine Telefonleitung verbunden sein.
Erweiterungseinschub	ein spezieller Verbindungsstecker innerhalb des Computers. Damit können zusätzliche Geräte angeschlossen werden. Siehe Kapitel 3.
Fehler	ein Problem, das den Computer veranlaßt, ein Programm falsch oder überhaupt nicht abzuarbeiten.
Fehlersuchen (engl. debugging)	Fehler in einem Programm entdecken, und/oder beseitigen.
Feld	eine Informationseinheit, die Teil einer Datei ist. Ein Beispiel: in der folgenden Adressliste sind NAME, STRASSE, STADT, LAND und PLZ Felder:

```
Adreßliste
NAME____________________
STRASSE_________________
PLZ_____
STADT___________________
LAND____________________
```

Fenster (Window)	der sichtbare Teil einer Bildschirmanzeige. Einige Computer gestatten die Eingabe von Zeilen mit 80 Zeichen, aber Sie sehen auf einmal nur ein Fenster mit 31 Zeichen. Elektronische Arbeitsblätter zeigen nur einen kleinen Ausschnitt ihres gesamten Arbeitsblatts.
Festplatte	ein Massenspeichergerät, das eine speziell beschichtete Metallplatte verwendet. Es übertrifft um ein Vielfaches die Speicherkapazität einer Diskette.
Feuerknopf	Knopf auf einem Joystick.
Firmware	Rechnerhardware mit einem gespeicherten Programm.
Floppy Disk	siehe Diskette.
Flußdiagramm	eine grafische Darstellung für die Struktur eines Programms.

Formatieren	das elektronische Organisieren einer Diskette, so daß Daten darauf abgespeichert und wieder geholt werden können.
Funktion	BASIC enthält eine Anzahl von eingebauten Funktionen. Jede Funktion hat einen Namen, gefolgt von Klammern. In den Klammern sind ein oder mehrere Argumente enthalten. Die Funktion hat einen einzelnen Wert (Zahl oder Zeichenkette), der durch das Argument bestimmt wird. Beispiele von Funktionen: ASC, CHR$, INT, LEN, RND, LEFT$, MID$ RIGHT$, STR$, VAL, TAB, PEEK
Funktionstasten	eine Taste, die den Computer veranlaßt, eine bestimmte Aktion durchzuführen. Im MSX-BASIC können sie frei zugeordnet werden.
Gerät	jedes Teil der Computerausrüstung.
GML	Grafische Makro-Sprache (Unterkommandos), die bei DRAW verwendet wird und MSX-BASIC Befehle gibt.
Grafik	Bilder, Strichzeichnungen, besondere Zeichen usw., die am Bildschirm oder auf einem Drucker dargestellt werden können.
Geschütztes Programm	im Speicher abgelegtes Programm, das wie ein Programm im ROM nicht gelöscht werden kann.
Halbleiter	ein Metall oder anderes Material (zum Beispiel Silizium), das leitende und nichtleitende Stellen besitzt. Sein elektrischer Widerstand kann durch Strom, Licht oder Hitze verändert werden.
Hard copy	Abdruck der Bildschirmdarstellung auf Papier.
Hardware	der mechanische Teil eines Rechners (der Computer selbst, der Drucker, die Tastatur und der Monitor).
Hertz	Einheit einer Übertragungssignal-Frequenz (Zyklen pro Sekunde)
Hexadezimales System	(richtig eigentlich: sedezimal) ein Zahlensystem, in dem jede Stelle eine Potenz zur Basis 16 bildet. Man braucht 16 Werte zur Darstellung der Zahlen (0-9, A-F).

Hochsprache	eine Programmiersprache, die Befehle und Anweisungen benutzt, die einer natürlichen Sprache ähneln.
IBM	Warenzeichen der International Business Machine Corporation.
Index	Ein anderes Wort für Index ist Kennziffer. Dem Namen einer Matrix folgen eine oder mehrere Zahlen oder numerische Variable in Klammern. Jede Zahl stellt einen Index dar.
Inhaltsverzeichnis	eine Liste aller Dateien auf einer Diskette.
Initialisieren	Ein Programmteil oder ein Hardware-Gerät wird in einen definierten Zustand gesetzt.
Integrierter Schaltkreis	eine Gruppe von Komponenten, die einen winzigen elektronischen Schaltkreis bilden. Alle Komponenten werden auf einem einzigen Stück Halbleiter hergestellt.
Interaktiv	Der Anwender steht im Dialog mit einem Computersystem.
Interpreter	Hochsprache, die permanent im Speicher eines Rechners vorhanden ist und ein Programm Anweisung für Anweisung ausführt. Programme, die mit einem Interpreter ausgeführt werden, sind langsamer als compilierte Programme, aber man findet Fehler schneller.
Interrupt	Die CPU unterbricht auf Anforderung ihren Programmablauf und bearbeitet einen anderen gewünschten Programmteil. Danach fährt sie mit der vorhergehenden Programmausführung fort.
Inversdarstellung	Die Darstellung der Zeichen am Bildschirm wird "umgekehrt" angezeigt, also dunkle Buchstaben auf hellem Hintergrund.
I/O	(Input/Output —> deutsch: Ein-/Ausgabe) Zuordnung zu einem Gerät oder Kanal, das die Ein- oder Ausgabe bedient.
Joystick	ein Gerät mit einem beweglichen Hebel und einem "Feuerknopf". Es wird häufig bei Computerspielen eingesetzt.

K
: eine Abkürzung für Kilobyte oder Kilobytes. In der Computerfachsprache steht 1K für 1024 Bytes.

Kassette
: eine kleine Plastikeinheit, die ein magnetisches Band enthält. Sie hat zwei Spulen, die über das Band miteinander verbunden sind. Computerprogramme können auf einer Kassette abgespeichert werden.

Kassetten-Recorder
: ein Recorder, auf dem Kassetten laufen und der zum Abspeichern von Computerprogrammen mit dem Rechner verbunden sein muß.

Kilobyte
: 1024 Bytes. 4 Kilobytes (abgekürzt: 4K Bytes oder 4K) entsprechen 4096 Bytes.

Kompatibilität
: die Fähigkeit, ein Programm unverändert auf verschiedenen Maschinen laufen zu lassen.

Konsole
: die Tastatur und andere Geräte, die die Steuereinheit eines Computers ausmachen.

Konstante
: eine Zahl oder eine Zeichenkette, die sich nicht verändert, während das Programm läuft. Ihr Wert wird direkt in der Programmzeile und nicht in einer Variablen gespeichert.

Laden
: Eine Datei (ein Programm) wird von einer Kassette oder Diskette in den Speicher des Computers geladen.

Lesen
: der Vorgang, Daten von einem Massenspeicher zu holen (zum Beispiel von einer Diskette) und sie in den Speicher des Computers abzulegen.

Logischer Operator
: ein Symbol, das den Computer veranlaßt, einen Vergleich durchzu führen. Es gibt folgende Operatoren: > (größer als), < (kleiner als) und = (ist gleich).

Maschinenprogramm
: Programm, dessen Anweisungen im Verhältnis 1:1 zu den Rechneranweisungen stehen. Diese Programme sind einem speziellen Mikroprozessor (wie 8085 oder 6502) zugeordnet und können nicht ohne weiteres auf andere Systeme angepaßt werden.

Massenspeicher
: zum Beispiel Diskettenlaufwerke und Kassetten-Recorder.

Matrix — ein Satz von Variablen, die den gleichen Namen haben. Die Elemente einer Matrix sind durchnumeriert. Die Zahlen erscheinen nach dem Variablennamen in Klammern. Beispiele:

```
A(0)        ist das erste Element der Matrix A
B$(7)       ist das achte Element der Matrix B$
CD(3,M+1)   ist ein Element der Matrix CD
```

Matrixdrucker — ein Drucker, der seine Zeichen mit Hilfe von Nadeln macht. Die Nadeln sind in einer Matrix angeordnet, zum Beispiel 5 * 7. Dieser Druckertyp wird am häufigsten eingesetzt.

Mega — bedeutet eine Million, wenn es einem Begriff vorangestellt ist.

Menü — eine Anzeige am Bildschirm, bei der Sie aus einer Liste die gewünschten Punkte auswählen können. In der Regel muß eine Zahl oder ein Buchstabe eingegeben werden.

Mikrocomputer — ein vollständig funktionsfähiger Computer, der einen Mikroprozessor als CPU besitzt.

Mikroprozessor — eine Zentraleinheit auf einem einzigen Siliziumchip.

Minicomputer — ein kleiner Rechner, der wie ein Computer der Groß-EDV aufgebaut ist.

MML — Musik-Makro-Sprache (Unterkommandos), die im MSX-BASIC-PLAY-Befehl verwendet wird.

Mnemonic — eine aus Buchstaben gebildete Symbolsprache, die es erleichtert, die Maschinensprache zu verwenden.

Modem — ein Gerät, das Daten so umformt, daß sie über eine Telefonleitung zu einem anderen Rechner (wieder über Modem) gesendet werden können.

Modul — eine Programmeinheit, die wie ein direkter Ausdruck getestet werden kann.

Modulares Programmieren — eine Technik zur Programmentwicklung, in dem ein Programm in logische Funktionen aufgeteilt wird, wobei jede Funktion ein einzelnes Modul ist.

Monitor	hat zwei Bedeutungen. Erstens bezeichnet das Wort ein besonderes Fernsehgerät, das mit dem Computer verbunden wird. Er zeigt Texte und Grafik, kann aber keine Fernsehprogramme empfangen. Zweitens bedeutet das Wort in der Maschinensprache: Kontrollprogramm.
Monitorprogramm	Software oder Firmware, die ein Betriebssystem kontrolliert.
MSX	Microsoft Software Exchange. Von Microsoft entwickelt.
Nano	ein Billionstel.
Nanosekunde	eine billionste Sekunde. Moderne Computer führen Befehle innerhalb weniger Nanosekunden aus.
Numerische Daten	Daten, die ausschließlich aus Zahlen bestehen.
Nullzeichenkette	eine Zeichenkette, die kein Zeichen enthält.
Objektdatei	ein Maschinenprogramm, das mit Assembler oder Compiler erstellt wurde.
Oktales System	Zahlensystem, in dem jede Stelle als Potenz zur Basis 8 dargestellt wird.
Parallele Schnittstelle	Verbindung zu einem Peripheriegerät, wobei die Daten im Gegensatz zur seriellen Schnittstelle immer als ein Byte geschickt werden.
PC	Personal-Computer
Peripheriegerät	jedes Gerät eines Rechnersystems, das außerhalb des Hauptprozessors ist, z.B. Drucker, Modem oder Terminal.
Pixel	die kleinste Einheit, die mit Farbe auf einem Bildschirm zu sehen ist.
Platine	eine Plastik- oder Fiberglaskarte, die Schaltkreise enthält.
Port	die Schnittstelle,an dem die Ein/Ausgabegeräte mit dem Computer verbunden sind.
Programm	eine Reihe von Instruktionen. Das Programm muß in einer Sprache geschrieben sein, die der Computer versteht.

Prozessor	Hauptteil eines Rechnersystems, das Befehle empfängt und ausführt.
Pseudocode	ein Anweisungscode, der noch übersetzt werden muß.
PSG	programmierbarer Tongenerator (Programmable Sound Generator).
Puffer	ein Bereich im Speicher zur vorübergehenden Speicherung von Informationen, die vom Computer ein- oder ausgegeben werden.
Quellprogramm	ein Programm, das nicht aus Maschinen-Code besteht. Es muß erst assembliert oder compiliert werden, damit es der Rechner versteht.
QWERTY	eine Abkürzung für eine Standardtastatur, wie sie im englischen Sprachraum verwendet wird. Die Kombination QWERTY geben die ersten sechs Buchstaben auf der linken Seite der Tastatur wieder.
RAM	englisch: Random Access Memory = wahlfreier Zugriffsspeicher. Die Daten können sowohl geschrieben als auch gelesen werden.
Register	Speicherstelle in der CPU oder im PSG, die spezielle Funktionen erfüllt.
RESET	Initialisierung des Systems. Erfolgt normalerweise beim Einschalten.
RGB-Monitor	Farbmonitor mit eigenen Eingängen für die Signale rot, grün und blau.
ROM	(Read-Only-Memory = nur-Lese-Speicher) Speicher, in dem Daten oder Programme permanent gespeichert sind und während des normalen Rechnerbetriebs nicht gelöscht werden können.
ROM-Modul	eine kleine Schachtel aus Plastik, die in ROMS gespeicherte Programme enthält, zum Beispiel BASIC. Meistens werden auf ROM-Modulen Spielprogramme angespeichert, die man dann einfach in den Heimcomputer steckt. Siehe Cartridge.

Schaltkreis	Die elektrische Schaltung eines Computers enthält Tausende von verschiedenen Bauteilen wie Transistoren, Dioden, Widerständen usw.
Schleife	ein Teil des Programms, der immer wieder ausgeführt wird.
Schnittstelle	eine Baugruppe, die es ermöglicht, das Geräte untereinander kommunizieren.
Schreib/Lesekopf	Bauteil des Diskettenlaufwerks, das Informationen von einer Diskette liest, schreibt oder löscht.
Schreib/Lesespeicher	siehe RAM.
Sedezimales System	(fälschlicherweise auch hexadezimal genannt) ein Zahlensystem, in dem jede Stelle eine Potenz zur Basis 16 bildet. Man braucht 16 Werte zur Darstellung der Zahlen (0-9, A-F).
Sektor	physikalische Speichereinheit auf einer Diskette.
Sequentielle Datei	Dateien, in denen die Zugriffszeit einer Speichereinheit (Datensatz) abhängig von seiner Entfernung vom Anfang der Datei ist.
Serielle Schnittstelle	Verbindung mit einem Peripheriegerät, wobei die Daten Bit-weise gesendet werden.
Silizium	ein nichtmetallisches chemisches Element. Es wird bei der Herstellung von Transistoren, Solarzellen usw. verwendet.
SLOT	siehe Kapitel 3.
Software	Programme, die auf dem Rechnersystem laufen.
Sprache	kommunizieren. Der Unterschied zwischen Computersprache und menschlicher Sprache liegt darin, daß es eine Computersprache ermöglicht, mit dem Menschen Verbindung aufzunehmen. Eine Hochsprache kann bereits von Leuten angewendet werden, die nichts oder nur wenig von Computern verstehen. Maschinensprachen setzen größeres Wissen voraus.

Speicher	der Teil des Computers, in dem Informationen aufbewahrt werden. Der Speicher besteht aus Halbleiterchips.
Speichern	Daten auf einem externen Datenträger wie Diskette oder Kassette ablegen.
Sprite	Siehe Kapitel 11.
Standardwerte	(default value) die Grundeinstellung eines Geräts oder eines Programms, wie sie vom Hersteller ausgeliefert wurde.
Statischer Speicher	ein Speicher, der Daten enthält, die nicht beständig aufgefrischt werden müssen (siehe auch Dynamischer Speicher).
Stromversorgung	eine Baugruppe, die einen Trafo und weitere Bauteile enthält. Sie transformiert die Spannung von 220V auf die Gleichspannung, die der Computer benötigt.
Strukturiertes Programmieren	Programmiertechnik, die mit einzelnen Programm-Modulen arbeitet.
Terminal	Gerät, das die Eingabe über Tastatur und Ausgabe auf den Bildschirm in Zusammenarbeit mit einem Rechner übernimmt.
Texteditor	ein Computerprogramm, mit dem Sie den Inhalt des Speichers modifizieren können. Man kann Daten oder Programme ändern.
Textverarbeitungssystem	ein Computerprogramm, mit dem man Text erstellen und ändern kann. Der Text kann formatiert ausgedruckt werden.
Tongeber	Der Computer erzeugt damit einen Ton (er piepst), um den Programmierer auf etwas hinzuweisen.
Typenraddrucker	eine druckende Maschine, deren Druckkopf aus einer Anzahl (gewöhnlich 96) kreisförmig angeordneter Stege besteht. Jeder Steg hat am Ende ein Zeichen. Typenraddrucker haben ein ähnliches (oder gleiches) Schriftbild wie Schreibmaschinen.
Urlader	Programm im ROM, das weitere Programme lädt.

Variable — eine Einheit, der man einen Namen gibt, und die einen beliebigen Wert annehmen kann. Ein Beispiel: A soll eine Variable mit dem Wert 1 sein. Addiert man 3 dazu, erhält die Variable A den neuen Wert 4.

Video-Display — der Bildschirm Ihres Fernsehgeräts oder Ihres Monitors.

Video-Display-Prozessor (VDP) — ein Mikroprozessor, der speziell für die Anzeige und Steuerung des Bildschirms entworfen wurde.

Wort — ein kleines Element im Speicher des Computers und die kleinste Dateneinheit, mit der die CPU arbeiten kann. Die Größe eines Wortes hängt von der Art des Computers ab. Sie kann von 8 Bits bis zu 12, 16, 32 oder 64 Bits betragen.

Z80 — ein 8-Bit-Mikrocomputer, der häufig in Personalcomputern eingesetzt wird.

Zeilennummer — eine Zahl, die jede Zeile eines Programms in einer Hochsprache definiert. Der Computer führt das Programm beginnend mit der niedrigsten Zeilennummer in aufsteigender Folge aus.

Zeilenorientierter Editorbefehl — Editorfunktion, die sich auf eine ganze Textzeile bezieht.

Zentraleinheit — das "Herz" des Computers. Sie besitzt die Fähigkeit, die Durchführung der Anweisungen zu steuern.

Zugriffszeit — die Zeitspanne, die vergeht, bis Daten von einer Diskette gelesen oder darauf geschrieben werden.

Anhang A: Zusammenfassung der MSX-BASIC-Befehle

Auf den nächsten Seiten finden Sie eine Zusammenfassung der MSX-BASIC-Befehle, -Anweisungen, -Funktionen und speziellen MSX-Variablen. Jeder Ausdruck wird kurz beschrieben.

A.1 Befehle

Befehle werden nach ihrer Eingabe sofort ausgeführt. Man setzt sie oft im direkten Modus ein. Mit Ausnahme von CONT können alle Befehle in einem Programm auch im indirekten Modus verwendet werden.

AUTO	ermöglicht automatisches Zeilennumerieren.
BLOAD	lädt Maschinensprachprogramme in den Speicher.
BSAVE	speichert Maschinensprachprogramme auf ein angegebenes Gerät.
CLEAR	Allen Zeichenketten-Variablen werden Nullzeichenketten zugewiesen und numerische Variablen erhalten den Wert Null.
CLOAD	lädt Programme vom Kassetten-Recorder in den Speicher.
CLOAD?	vergleicht ein Programm auf Kassette mit dem im Speicher.
CONT	führt die Ausführung fort.
CSAVE	speichert ein Programm auf Kassette.
DELETE	löscht Programmzeilen aus dem Speicher.
LIST	zeigt das ganze Programm oder einen Teil davon auf dem Bildschirm an. Das Programm muß im Speicher stehen.
LLIST	gibt das ganze Programm oder einen Teil davon auf den Drucker aus. Das Programm muß im Speicher stehen.
LOAD	lädt eine ASCII-Programmdatei von einem angegebenen Gerät in den Speicher.
MERGE	hängt eine ASCII-Programmdatei von einem Gerät an ein Programm im Speicher an.

NEW	löscht den gesamten Speicher.
RENUM	numeriert die Programmzeilen neu.
RUN	führt das Programm aus, das im Speicher steht.
SAVE	speichert ein ASCII-Programm auf ein angegebenes Gerät.
TRON/TROFF	schaltet den Fehlersuchmodus ein/aus.

A.2 Anweisungen

Die Anweisungen in MSX-BASIC können in acht funktionale Gruppen eingeteilt werden: Datentyp-Definitionen, Zuweisungen und Zuordnungen, Steuerungs-, bedingte Ausführungs-, Nicht-Ein/Ausgabe-, Ein/Ausgabe-, Farb- und Grafik-, Sound-, Zeit- und Interrupt-Anweisungen. In der nachfolgenden Aufzählung sind sie ihrer Funktion nach aufgeführt.

A.2.1 Datentyp-Definitionen

DEFDBL	definiert Variablen mit doppelter Genauigkeit.
DEFINT	definiert Ganzzahl-Variablen.
DEFSNG	definiert Variablen mit einfacher Genauigkeit.
DEFSTR	definiert Zeichenketten-Variablen.

A.2.2 Zuweisungen und Zuordnungen

DIM	reserviert Speicherplatz für Matrizen.
ERASE	löscht Matrizen aus dem Programm.
LET	ordnet einer Variablen einen Wert zu.
REM	Damit werden Bemerkungen gemacht.
SWAP	vertauscht Variablen-Werte.

A.2.3 Steueranweisungen

END	beendet das Programm.
FOR/NEXT	führt eine Reihe von Anweisungen innerhalb einer Schleife durch. Die Schleife ist durch eine gegebene Zahl begrenzt.
GOSUB/RETURN	springt in ein Unterprogramm und kehrt von dort zurück.
GOTO	unkonditioniertes Springen an eine angegebene Zeile.
ON ERROR GOTO	ermöglicht eine Fehlersuch-Routine.
ON GOTO/ON GOSUB	springt, abhängig von einem gegebenen Ausdruck, zu einer der angegebenen Zeilen.
ON INTERVAL GOSUB	springt nach einem angegebenen Zeitinterval zu der gesetzten Zeile. Wird mit INTERVAL ON/OFF/STOP verwendet.
ON KEY GOSUB	geht in ein entsprechendes Unterprogramm, wenn die angegebene Funktionstaste gedrückt wurde.
ON SPRITE GOSUB	springt in ein Unterprogramm, wenn Sprites zusammenstoßen. Wird mit SPRITE ON/OFF/STOP verwendet.
ON STOP GOSUB	springt in ein Unterprogramm, wenn <CTRL>+<STOP> gedrückt wurden. Wird mit STOP ON/OFF/STOP verwendet.
ON STRIG GOSUB	springt in ein Unterprogramm, wenn der Joystick-Knopf (bzw. die Leertaste) gedrückt wurde. Wird mit STRIG ON/OFF/STOP verwendet.
RESUME	führt nach der Rückkehr aus einer Fehlersuch-Routine die Ausführung weiter fort.
RETURN	Rückkehr aus einem Unterprogramm.
STOP	beendet die Ausführung und kehrt in den Befehlsmodus zurück.

A.2.4 Bedingte Ausführungsanweisungen

IF/THEN/ELSE	abhängig vom Ergebnis eines Ausdrucks springt das Programm zu der entsprechenden Zeile.

A.2.5 Nicht-Ein/Ausgabe-Anweisungen

CALL	ruft eine erweiterte Anweisung von einem SLOT auf.
DATA	speichert Zahlen und Zeichenketten-Konstanten.
DEF FN	definiert Funktionen.
DEFUSR	definiert die Startadresse eines Maschinensprach-Unterprogramms.
ERROR	simuliert einen Fehler.
KEY	ordnet eine Zeichenkette einer angegebenen Funktionstaste zu.
KEY LIST	zeigt die momentane Zuordnung der Funktionstasten an.
KEY ON/OFF	schaltet die Funktionstastenanzeige ein oder aus.
LOCATE	bewegt den Cursor zu einer angegebenen Position auf dem Bildschirm.
MID$	ersetzt einen Teil einer Zeichenkette mit einer anderen.
READ	liest Daten von einer DATA-Anweisung und ordnet sie einer angegebenen Variablen zu..
RESTORE	setzt den DATA-Zeiger zurück, so daß die DATA-Anweisungen von Anfang an wieder gelesen werden können.

A.2.6 Ein/Ausgabe-Anweisungen

BEEP	sendet einen Piepston zum Lautsprecher.
CLOSE	schließt die Ein/Ausgabekanäle und gibt den zugeordneten Puffer frei.
CLS	löscht den Bildschirm.
INP	gibt die Eingabe eines Ports zurück.
INPUT	erlaubt während der Programmausführung die Eingabe über die Tastatur.
INPUT#	liest sequentielle Daten von einem angegebenen Kanal.

INTERVAL ON/OFF/STOP gibt abhängig von einem Zeitinterval den Ablauf eines Programmteils frei oder sperrt ihn.

KEY<n> ON/OFF/STOP gibt abhängig von einer angegebenen Funktionstaste den Ablauf eines Programmteils frei oder sperrt ihn.

LINE INPUT erlaubt die Eingabe einer ganzen Zeile (bis zu 254 Zeichen). Die Eingabe wird einer Zeichenketten-Variablen zugeordnet.

LINE INPUT# liest die gesamte Zeile von einer sequentiellen Datei.

LPRINT gibt Daten auf dem Drucker aus.

LPRINT USING gibt die Daten in einem angegebenen Format auf einen Drucker aus.

MOTOR ON/OFF schaltet den Kassetten-Motor ein oder aus.

OPEN ordnet einen Puffer zu und setzt den Ein/Ausgabe-Modus

OUT sendet ein Byte an ein Ausgabeport.

PRINT zeigt Daten auf dem Bildschirm an.

PRINT USING zeigt Daten in einem angegebenen Format auf dem Bildschirm an.

PRINT# schreibt Daten in eine sequentielle Datei.

PRINT# USING schreibt Daten in einem angegebenen Format in eine sequentielle Datei.

SPRITE ON/OFF/STOP gibt die Abfrage nach einer Sprites-Kollision frei oder sperrt sie.

STOP ON/OFF/STOP gibt die Abfrage, ob die Tasten <CTRL>+<STOP> gedrückt sind, frei oder sperrt sie.

WAIT unterbricht die Ausführung.

STRIG ON/OFF/STOP gibt die Abfrage, ob der Joystick-Knopf gedrückt wurde, frei oder sperrt sie.

WIDTH setzt im Textmodus die Zahl der Zeichen pro Zeile.

A.2.8 Farb- und Grafikanweisungen

COLOR	wählt den Vordergrund, den Hintergrund und die Rahmenfarbe für die Anzeige am Bildschirm.
CIRCLE	zeichnet eine Ellipse (oder Kreis).
DRAW	zeichnet mit Hilfe der Grafik-Makrosprache Bilder den Bildschirm.
LINE	zieht Linien und Rechtecke auf dem Bildschirm.
PAINT	füllt umrahmte Figuren mit einer angegebene Farbe.
PRESET	löscht einen Punkt an einer angegebenen Stelle des Bildschirms aus.
PSET	zeichnet einen Punkt an eine angegebene Stelle des Bildschirms.
PUTSPRITE	setzt die Sprite-Attribute. Gibt das Bild, das in ein Matrix abgelegt ist, auf dem Bildschirm aus.
SCREEN	wählt einen der verschiedenen Bildschirm-Modi, die Sprite-Größe, den Key-Klick, die Kassetten-Baudrate und die Druckeroptionen aus.

A.2.8 Sound-Anweisungen

PLAY	spielt Musik, die über die Musik-Makro-Sprache eingegeben wird.
SOUND	schreibt direkt in die Register des PSG Werte ein.

A.3 Funktionen

MSX-BASIC hält eine Reihe von eingebauten Funktionen bereit. Man kann sie in drei Kategorien unterteilen: arithmetische, Zeichenketten- und spezielle Funktionen.

A.3.1 Arithmetische Funktionen

ABS	gibt den absoluten Wert zurück.

ATN	gibt den Arcustangens zurück.
CDBL	wandelt in doppelte Genauigkeit um.
CINT	wandelt in Ganzzahlen um.
COS	gibt den Cosinus in Radiant zurück.
CSNG	wandelt in einfache Genauigkeit um.
EXP	gibt den Exponentialwert zurück.
FIX	gibt den Ganzzahl-Anteil des Arguments zurück.
INT	gibt die größte Ganzzahl, die kleiner oder gleich dem Argument ist, zurück.
LOG	gibt den natürlichen Logarithmus zurück.
RND	gibt eine Zufallszahl zwischen 0 und 1 zurück.
SGN	gibt das Vorzeichen (+ oder -) eines Arguments zurück.
SIN	gibt den Sinus in Radiant zurück.
SQR	gibt die Wurzel zurück.
TAN	gibt den Tangens zurück.

A.3.2 Zeichenketten-Funktionen

ASC	gibt einen numerischen Wert zurück, der dem ASCII- Code des ersten Zeichens des String-Arguments entspricht.
BIN$	gibt eine Zeichenkette zurück, die dem binären Wert des dezimalen Arguments entspricht.
CHR$	gibt eine Zeichenkette mit einem Element zurück, dessen Argument ASCII-Code enthält.
EOF	gibt -1 (wahr) zurück, wenn das Ende einer sequentiellen Datei erreicht wurde.
HEX$	gibt eine Zeichenkette zurück, die dem hexadezimalen Wert des dezimalen Arguments entspricht.

INPUT$	gibt eine Zeichenkette zurück, die von der Tastatur gelesen wurde.
INSTR$	sucht nach einer Teilzeichenkette in einer Zeichenkette.
LEFT$	gibt die angegebenen Zeichen links einer Zeichenkette zurück.
LEN	gibt die Länge einer Zeichenkette zurück.
MID$	gibt einen Teilstring einer Zeichenkette zurück.
OCT$	wandelt einen Dezimalwert in einen Oktalwert um.
RIGHT$	gibt den rechten Teil einer Zeichenkette zurück.
SPACE$	gibt eine Zeichenkette aus Leerzeichen zurück.
STR$	wandelt eine Zahl in eine Zeichenkette um.
STRING$	bildet eine Zeichenkette.
VAL	gibt den numerischen Wert einer Zeichenkette zurück.

A.3.3 Spezielle Funktionen

CSRLEN	gibt die momentane Zeilenposition des Cursors zurück.
FRE	gibt die Anzahl der Bytes zurück, die von MSX-BASIC noch nicht benutzt wurden.
LPOS	gibt die Position des Druckerkopfes zurück.
MAXFILES	gibt die maximale Zahl der Dateien, die mit einmal geöffnet sind, an.
PAD	gibt den Status eines Tabletts zurück.
PDL	gibt den Wert eines Paddles zurück.
PEEK	liest das Byte einer angegebenen Speicherstelle.
POINT	gibt die Farbe eines angegebenen Pixels am Bildschirm zurück.
POKE	schreibt ein Byte in eine angegebene Speicherstelle.

POS	gibt die momentane Spaltenposition des Cursors am Bildschirm zurück.
SPC	druckt Leerräume am Bildschirm oder am Drucker aus.
STICK	gibt die Richtung eines Joysticks an.
STRIG	gibt den Status des Joystick-Knopfes an.
TAB	druckt Leerräume bis zu einer bestimmten Position am Bildschirm oder am Drucker.
USR	ruft ein Maschinensprach-Programm auf.
VARPTR	gibt die Adresse einer Variablen zurück, die im Speicher abgelegt ist.
VPEEK	liest den Wert der angegebenen Speicherstelle im VRAM.
VPOKE	schreibt den Wert in eine angegebene Speicherstelle im VRAM.

A.4 Spezielle Variablen

Anschließend finden Sie eine Liste der speziellen Variablen von MSX-BASIC.

BASE	setzt die momentane Basisadresse für verschiedene Video-Display-Tabellen.
ERR und ERL	geben den Fehlercode und die Zeilennummer, in der ein Fehler passiert, zurück.
INKEY$	gibt ein Zeichen zurück, wenn die Tastatur betätigt wurde oder 0, wenn keine Taste gedrückt wurde.
SPRITE$	setzt die Muster eines Sprite.
TIME	setzt den Systemzähler des VDP, damit Interrupts generiert werden können (60mal pro Sekunde).
VDP	gibt den momentanen Wert des Schreib- oder Leseregisters des VDP an.

Anhang B: MSX-BASIC-Fehlercodes und -Meldungen

Das Fehlersuchen ist ein unvermeidlicher Teil des Programmierens. Erkennt das MSX-BASIC einen Fehler, zeigt es eine Fehlermeldung am Bildschirm an. Lesen Sie diese Fehlermeldung und sehen Sie, wenn es nötig ist, in der kurzen Erklärung in diesem Anhang nach. MSX-BASIC hilft Ihnen, die Programme fehlerfrei zu machen. Die erste Zahl gibt jeweils den Fehlercode an.

1. NEXT without FOR (Next ohne FOR)

Der Computer hat einen NEXT-Befehl erreicht und das dazugehörende FOR nicht gefunden. Unter Umständen ist NEXT eine falsche Variable zugeordnet worden. Beispiel:

```
10 FOR I=1 TO 7
20 PRINT I
30 NEXT X               (Es sollte NEXT I heißen.)
```

2. Syntax error (Syntaxfehler)

In einer Zeile wurde ein Ausdruck (Befehl) falsch geschrieben. Beispiele:

```
(a)      10 A)=8

(b)      10 DEM A(2,2,2)    (Es sollte DIM heißen.)
```

Es kann auch sein, daß die Daten in einer DATA-Anweisung nicht mit dem Variablentyp übereinstimmen. Beispiel:

```
10 READ A
20 DATA "HALLO"
```

3. RETURN without GOSUB (RETURN ohne GOSUB)

Der Computer stößt auf eine RETURN-Anweisung, bevor er über einen GOSUB-Befehl kommt. Beispiel:

```
10 GOSUB 100
20 REM Hier beginnt das Hauptprogramm
80 REM
81 REM
100 REM UNTERPROGRAMM
110 REM Hier steht das Unterprogramm
199 RETURN
```

Das Programm kann verbessert werden, wenn man eine Zeile hinzufügt:

```
99 END
```

4. Out of data (keine Daten mehr)

READ wurde so oft verwendet, daß keine Daten in einer DATA-Anweisung mehr vorhanden sind. Beispiel:

```
10 FOR I=1 TO 4
20 READ N$
30 NEXT I
99 DATA THOMAS,MARKUS,MELANIE
```

5. Illegal function call (falscher Funktionsaufruf)

Diese Meldung kann die unterschiedlichsten Ursachen haben.

- Es wurde eine falsch dimensionierte Matrix eingegeben:

```
10 DIM A(-5)
```

- Der Befehl SQR wurde mit einem negativen Argument angegeben.
- Der Befehl LOG wurde mit einem negativen Argument oder mit 0 angegeben.

6. Overflow (Zahl zu groß)

Die Zahl, die berechnet werden sollte, übersteigt das zulässige Format.

```
10 X=2
20 FOR I=1 TO 50
30 X=X*X
40 PRINT X
50 NEXT I
```

7. Out of memory (Speicherbereich zu klein)

- Ein Programm ist zu groß.
- Es wurden zu viele FOR- oder GOSUB-Schleifen verwendet.
- Es wurden zu viele Variablen oder Ausdrücke eingesetzt.

8. Undefined line number (nicht definierbare Zeilennummer)

- Eine Zeilennummer wurde mit GOTO, GOSUB, RESTORE usw. angesprochen, die es im Programm gar nicht gibt.

9. Subscript out of range (Index nicht erlaubt)

Es wurde eine Matrix mit einem Index verwendet, der negativ oder zu groß ist.

(a)
```
10 A(-3)=5
```

(b)
```
10 DIM B(8)
20 B(99)=4
```

10. Redimensioned array (Matrix nochmals definiert)

Eine Matrix wurde mit zwei DIM-Anweisungen definiert oder eine Variable, die bereits benutzt wird, wird mit der DIM-Anweisung nochmals angesprochen.

11. Division by zero (Division durch Null)

Der Computer sollte eine Zahl durch Null dividieren, oder durch eine Variable, deren Wert Null ist.

```
10 A=0
20 B=7/A
```

12. Illegal direct (falsche direkte Eingabe)

Es wurde versucht, dem Computer einen Befehl ohne Zeilennummer einzugeben. Die meisten Befehle sind im direkten Modus gültig, aber einige bilden eine Ausnahme:

```
DEF FNA(X)=X*X
```

Die DEF FN-Anweisung darf nur in einer numerierten Zeile stehen.

13. Type mismatch (falsche Zuordnung)

Eine Zahl wurde einer Zeichenkettenvariablen zugeordnet oder umgekehrt.

(a)
```
A="HALLO"
```

(b)
```
B$=D
```

14. Out of string space (zu wenig Platz)

Es wurden so viele Zeichenketten eingesetzt, daß der Computer nicht genügend Speicher mehr frei hat.

15. String too long (Zeichenkette zu lang)

Jede Zeichenkette darf maximal aus 255 Zeichen bestehen.

```
10 A$="HALLO"
20 A$=A$+A$
30 PRINT A$
40 GOTO 20
```

16. String formula too complex (String zu kompliziert)

Die angegebene Zeichenkettenanweisung ist zu kompliziert. Sie muß in mehrere Abschnitte aufgeteilt werden.

17. Can't continue (Programm nicht weiter ausführbar)

BASIC kann das Programm nicht weiter abarbeiten, wenn es:

- nicht existiert.
- mit einer Fehlermeldung angehalten hat.
- nach der Unterbrechung geändert wurde.

18. Undefined user function (undefinierte Anwenderfunktion)

Eine Anwenderfunktion (FN) wurde aufgerufen, bevor sie mit DEF FN definiert wurde.

19. Device I/O Error (Ein/Ausgabefehler)

Es wurde beim Schreiben oder Lesen auf oder vom Kassetten- Recorder, Drucker oder Bildschirm ein Ein/Ausgabefehler entdeckt.

20. Verify error (Fehler beim Dateivergleich)

Das Programm im Speicher stimmt mit dem auf der Kassette nicht überein.

21. No RESUME (RESUME fehlt)

Ein Fehler wurde nicht durch eine RESUME-Anweisung behoben.

22. RESUME without ERROR (RESUME ohne ERROR)

Die RESUME-Anweisung muß nach einer ON ERROR GOTO-Anweisung stehen.

23. Unprintable error (kein zuordenbarer Fehler)

Es gibt für den Fehler keine spezielle Fehlermeldung. Diese Meldung wird in aller Regel ausgegeben, wenn ein Fehler mit einem undefinerten Code auftritt. Der Anwender kann hier eigene "Fehler" festlegen.

24. Missing operand (fehlender Operand)

Ein Teil einer mathematischen Anweisung wurde vergessen:

```
10 A=3*

20 B$="HALLO"+
```

25. Line buffer overflow (Zeilenpufferüberlauf)

Es wurde eine Zeile eingegeben, die zu viele Zeichen besitzt (>255).

26...49. Unprintable error (kein zuordenbarer Fehler)

Zur Erweiterung des MSX-BASIC vorgesehen. Der Anwender kann hier eigene "Fehler" festlegen.

50. FIELD overflow (Feldüberlauf)

Die Gesamtzahl der Bytes ist größer als die in der OPEN- Anweisung festgelegt Datensatzlänge.

51. Internal error (Interner Fehler)

Der Computer kann das Programm nicht abarbeiten. Entweder ist das Programm, das geladen wurde, nicht in Ordnung oder die Hardware des Computers hat einen Fehler.

52. Bad file number (falsche Dateinummer)

Die Datei, die mit der Dateinummer angesprochen wird, wurde noch nicht mit OPEN eröffnet oder die Nummer übersteigt die durch MAXFILES angegebene höchste Dateinummer.

53. File not found (Datei nicht gefunden)

Der Computer kann die angesprochene Datei nicht finden.

54. File already open (Datei schon eröffnet)

- Ein OPEN-Befehl wurde auf eine Datei angewendet, die schon eröffnet war.

- Eine eröffnete Datei soll mit KILL gelöscht werden.

55. Input past end (Eingabe nach Ende)

Es wurde versucht, mit INPUT Daten von einer Datei zu lesen, die bereits vollständig gelesen worden war.

56. Bad file name (falscher Dateiname)

Es wurde ein falscher Dateiname mit LOAD, SAVE, KILL usw. verwendet.

57. Direct statement in file (direkte Anweisung in der Datei)

Es wurde ein direkter Ausdruck beim Laden einer ASCII-Datei gefunden.

58. Sequential I/O only (nur sequentielle Ein/Ausgabe zugelassen)

Es wurde versucht, direkt eine sequentielle Datei zu lesen oder zu beschreiben.

59. File not open (Datei nicht eröffnet)

Die mit INPUT#, PRINT# usw. angesprochene Datei ist noch nicht eröffnet.

60...255. Unprintable error (kein zuordenbarer Fehler)

Die Fehlercodes sind nicht definiert. Der Anwender kann hier eigene "Fehler" festlegen.

Anhang C: Zahlensysteme und ASCII-Code

Wenn man genauer betrachtet, was ein Computer tut, sieht man, daß er nur zwei Sachen kennt: den Zustand ein und aus. Der Computer versteht nur Daten, die im Zweier- oder Binärsystem dargestellt sind (1 und 0). Wir wollen dieses und andere Zahlensysteme mit dem Dezimalsystem, das wir täglich benutzen, in Beziehung bringen.

C.1 Binäres Zahlensystem

Einige Mikroprozessoren verwenden einen 8-Bit-Datenbus, über den sie mit dem System kommunizieren. Wir definieren Bit 0 als das niederwertigste Bit und numerieren die Bits in aufsteigender Reihenfolge von 0 bis 7. Bit 7 wird das höherwertigste Bit genannt. Jedes Bit kann entweder ein (dargestellt durch 1) oder aus (dargestellt durch 0) sein. In einem 8-Bit-Datenwort gibt es 256 Möglichkeiten (2^8). Die Darstellungsweise sieht folgendermaßen aus:

Position	7	6	5	4	3	2	1	0
Exp.darstellg.	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
binär	0	0	1	0	0	0	0	1
dezimal	128	64	32	16	8	4	2	1

Die Dezimalwerte sind angegeben, um die Beziehung deutlicher zu machen. In unserem Beispiel sind die Bits in den Positionen 0 und 5 gesetzt. Das bedeutet, daß die Dezimalzahlen 1 und 32 relevant sind. Die Summe ist 33 (0*128 + 0*64 + 1*32 + 0*16 + 0*8 + 0*4 + 0*2 + 1*1). Die Binärdarstellung von 00100001 entspricht also dem Dezimalwert 33.

Werden im MSX-BASIC binäre Konstanten verwendet, müssen sie den Präfix "&B" davor stehen haben. Zum Beispiel: &B01110110 (=118).

C.2 Das oktale Zahlensystem

Das oktale Zahlensystem (Basis 8) teilt Binärstellen in Felder von 3 Bits. Begonnen wird mit den niedrigsten Datenbit (0). Die ersten drei Bits stellen die acht Zahlen (0 bis 7) folgendermaßen dar:

binär	oktal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Damit Zahlen, die größer als 7 sind, dargestellt werden können, müssen wir ein weiteres Feld mit drei Binärstellen anordnen:

binär	oktal	dezimal
000 000	1 0	8
001 001	1 1	9
... ...	. .	.
... ...	. .	.
... ...	. .	.
111 111	7 7	63

Um eine Zahl, die größer als 63 ist, darzustellen, müssen wir ein drittes Feld anfügen. Bei einem 8-Bit-Datenwort führt das dazu, daß das dritte Feld nur aus zwei Bits besteht, so daß die größte Oktalzahl, die im dritten Feld möglich ist, die Zahl 3 ist.

binär	oktal	dezimal
11 111 111	3 7 7	255

Der größte Vorteil des oktalen Systems gegenüber dem binären liegt darin, daß die gleiche Information in drei Stellen und nicht in acht Stellen dargestellt werden kann. Werden oktale Konstanten in MSX-BASIC verwendet, muß ihnen der Präfix "&O" voranstehen. Ein Beispiel: &O377 (=255).

C.3 Das hexadezimale Zahlensystem

Das sedezimale System (Basis 16), oft auch hexadezimales oder Hexsystem genannt, verwendet für jede Zahl zwei Felder mit jeweils vier binären Stellen. Die ersten vier Bits stellen die 16 Zahlen von 0 bis 15 dar. Damit man nur ein einziges Zeichen für die Zahlen von 10 bis 15 hernehme muß, werden die Buchstaben A bis F eingesetzt:

binär	**hexadezimal**	**dezimal**
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Damit eine Zahl größer als 15 dargestellt werden kann, müssen wir ein weiteres Feld aus vier binären Stellen anfügen:

Binär	**hexadezimal**	**dezimal**
0001 1111	1 F	16
....	. .	..
....	. .	..
....	. .	..
1111 1111	F F	255

Der Vorteil der hexadezimalen Zahlen besteht darin, daß nur noch zwei Stellen für die Darstellung von 8 Binärstellen verwendet werden müssen. Da Mikrocomputer Adressen und Datenfelder verwenden, die ein vielfaches von vier Bits betragen, ist die zahlenmäßige Darstellung im sedezimalen System am einfachsten.

Hexkonstanten im MSX-BASIC müssen mit dem Präfix "&H" gekennzeichnet sein. Ein Beispiel: &HFF (=255).

C.4 ASCII-Code

Der ASCII-Zeichensatz verwendet 7 Datenbits (0 bis 6). Das höchste Bit (Bit 7) wird gewöhnlich ignoriert oder weggeworfen. Anschließend finden Sie eine Tabelle mit den ASCII-Codes (= American Standard for Coded Information Interchange).

Control-Zeichen werden durch ein ^-Zeichen abgekürzt; Control C sieht dann so aus: ^C.

Dezimal	Hex	Oktal	Binär	ASCII
0	00	000	00000000	Null (NUL)
1	01	001	00000001	^A (SOH)
2	01	002	00000010	^B (STX)
3	03	003	00000011	^C (ETX)
4	04	004	00000100	^D EOT
5	05	005	00000101	^E ENQ
6	06	006	00000110	^F ACK
7	07	007	00000111	^G (Tongeber) BEL
8	08	010	00001000	^H (backspace) BS
9	09	011	00001001	^I (TAB) Hor. HT
10	0A	012	00001010	^J (Zeilenvor.) LF
11	0B	013	00001011	^K Vert. VT
12	0C	014	00001100	^L (Seitenvor.) FF
13	0D	015	00001101	^M (Wagenrückl.) CR Enter
14	0E	016	00001110	^N SO
15	0F	017	00001111	^O SI
16	10	020	00010000	^P DLE
17	11	021	00010001	^Q DC1
18	12	022	00010010	^R DC2
19	13	023	00010011	^S DC3
20	14	024	00010100	^T DC4
21	15	025	00010101	^U NAK
22	16	026	00010110	^V SYN
23	17	027	00010111	^W ETB
24	18	030	00011000	^X CAN
25	19	031	00011001	^Y EM
26	1A	032	00011010	^Z SUB
27	1B	033	00011011	Escape
28	1C	034	00011100	FS
29	1D	035	00011101	GS
30	1E	036	00011110	RS
31	1F	037	00011111	US
32	20	040	00100000	Leerraum
33	21	041	00100001	!
34	22	042	00100010	"
35	23	043	00100011	#
36	24	044	00100100	$
37	25	045	00100101	%
38	26	046	00100110	&
39	27	047	00100111	'
40	28	050	00101000	(

Dezimal	Hex	Oktal	Binär	ASCII
41	29	051	00101001	)
42	2A	052	00101010	*
43	2B	053	00101011	+
44	2C	054	00101100	,
45	2D	055	00101101	-
46	2E	056	00101110	.
47	2F	057	00101111	/
48	30	060	00110000	0
49	31	061	00110001	1
50	32	062	00110010	2
51	33	063	00110011	3
52	34	064	00110100	4
53	35	065	00110101	5
54	36	066	00110110	6
55	37	067	00110111	7
56	38	070	00111000	8
57	39	071	00111001	9
58	3A	072	00111010	:
59	3B	073	00111011	;
60	3C	074	00111100	<
61	3D	075	00111101	=
62	3E	076	00111110	>
63	3F	077	00111111	?
64	40	100	01000000	@ deutsch §
65	41	101	01000001	A
66	42	102	01000010	B
67	43	103	01000011	C
68	44	104	01000100	D
69	45	105	01000101	E
70	46	106	01000110	F
71	47	107	01000111	G
72	48	110	01001000	H
73	49	111	01001001	I
74	4A	112	01001010	J
75	4B	113	01001011	K
76	4C	114	01001100	L
77	4D	115	01001101	M
78	4E	116	01001110	N
79	4F	117	01001111	O
80	50	120	01010000	P
81	51	121	01010001	Q
82	52	122	01010010	R
83	53	123	01010011	S
84	54	124	01010100	T

Dezimal	Hex	Oktal	Binär	ASCII
85	55	125	01010101	U
86	56	126	01010110	V
87	57	127	01010111	W
88	58	130	01011000	X
89	59	131	01011001	Y
90	5A	132	01011010	Z
91	5B	133	01011011	[deutsch Ä
92	5C	134	01011100	\ deutsch Ö
93	5D	135	01011101	] deutsch Ü
94	5E	136	01011110	^
95	5F	137	01011111	_
96	60	140	01100000	`
97	61	141	01100001	a
98	62	142	01100010	b
99	63	143	01100011	c
100	64	144	01100100	d
101	65	145	01100101	e
102	66	146	01100110	f
103	67	147	01100111	g
104	68	150	01101000	h
105	69	151	01101001	i
106	6A	152	01101010	j
107	6B	153	01101011	k
108	6C	154	01101100	l
109	6D	155	01101101	m
110	6E	156	01101110	n
111	6F	157	01101111	o
112	70	160	01110000	p
113	71	161	01110001	q
114	72	162	01110010	r
115	73	163	01110011	s
116	74	164	01110100	t
117	75	165	01110101	u
118	76	166	01110110	v
119	77	167	01110111	w
120	78	170	01111000	x
121	79	171	01111001	y
122	7A	172	01111010	z
123	7B	173	01111011	{ deutsch ä
124	7C	174	01111100	| deutsch ö
125	7D	175	01111101	} deutsch ü
126	7E	176	01111110	~ deutsch ß
127	7F	177	01111111	Delete, Rubout

Anhang D: Das Inhaltsverzeichnis der Kassette

Die Programmbeispielkassette, die Sie beim Verlag erwerben können, enthält alle Programm-Module, die wir in diesem Buch beschrieben haben. Damit Sie sie nicht aus Versehen löschen, haben wir sie schreibgeschützt. Für Ihre eigene Arbeit brauchen Sie deshalb eine eigene Kassette.

Seite A	Seite B	
SZENE1	PLNT32	*
SZNE2M	PLNT16	*
SZENE2	AUTO1	*
SZNE2M	AUTOM	
SZENE3	AUTO2	*
SZNE4M	MALPRG	*
SZENE5		
SZENE5		
SZNE6M		
SZENE6		
SZENE7		
SCHUSS		
SCHU2M		
SCHUS2		
SCHU3M		
SCHUS3		

* Die Programme, die mit einem Sternchen markiert sind, sind gewöhnliche BASIC-Programm-Dateien und werden mit dem CLOAD-Befehl geladen. Die anderen sind ASCII-Dateien und können mit LOAD "CAS: oder MERGE "CAS: in den Speicher geholt werden.

Literaturverzeichnis

Coniffe, Patricia "Dictionary of Computer Terms Made Simple", Scholastic Inc. 1984

"MSX Technical Book Vol. 1 & 2", ASCII-Corporation 1983

Naisbitt, John "Megatrends Ten New Directions Transforming Our Lives", Warner Books, Inc. 1982

"Programmable Sound Generator Data Manual", General Instrument Corporation, 1981

Smith, Brian R. und Watts, Lisa "Usborne Guide to better Basic", EDC Publishing, 1983

Smith, Brian R. und Watts, Lisa "Usborne Guide to Computer's", Hayes Book, 1981

Stockley, Corinne und Watts, Lisa "Usborne Guide to Computer Jargon", EDC Publishing, 1983

"TMS 9918A Video Display Processor Data Manual", Texas Instruments Inc., 1980



Stichwortverzeichnis

